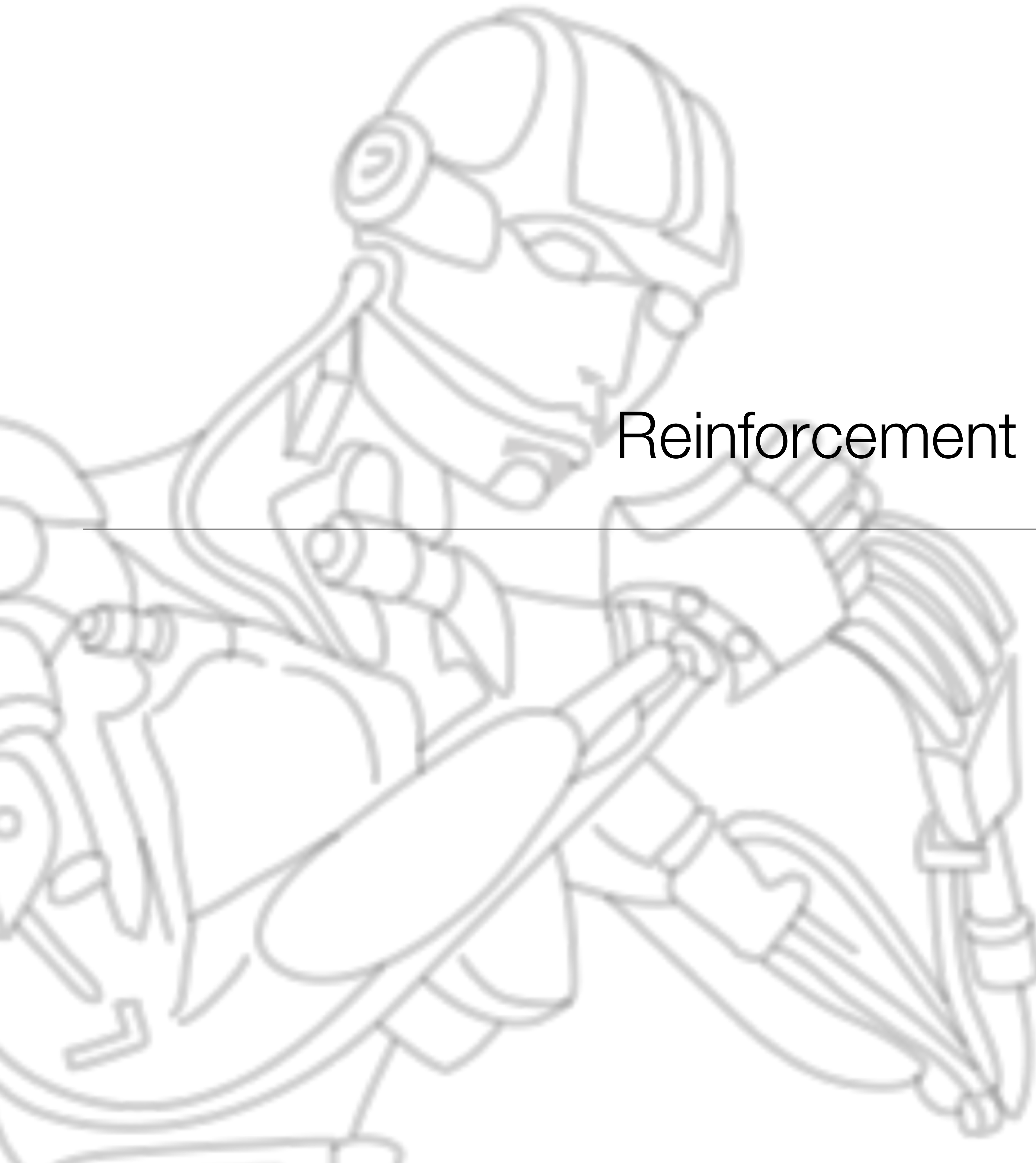
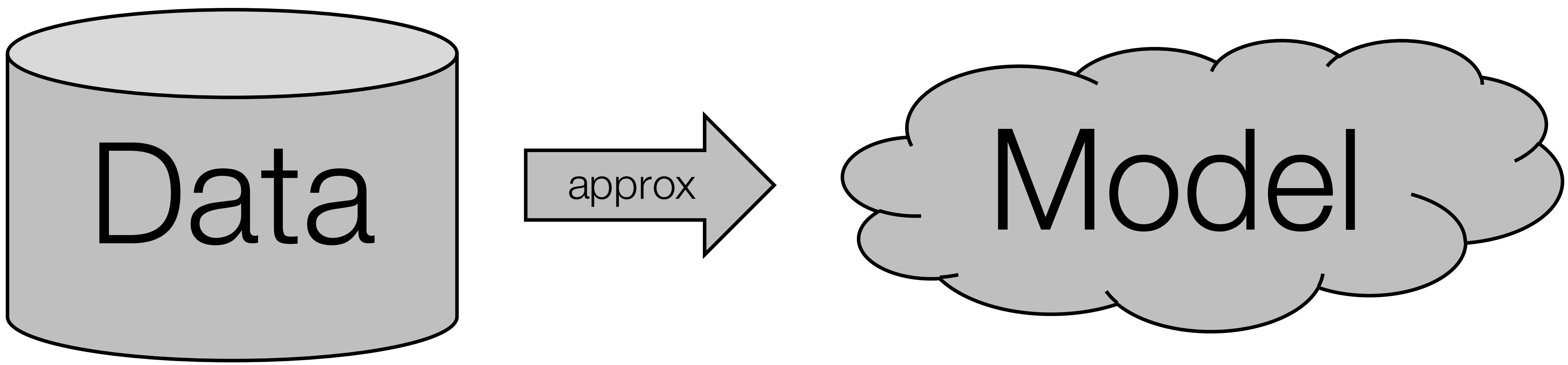


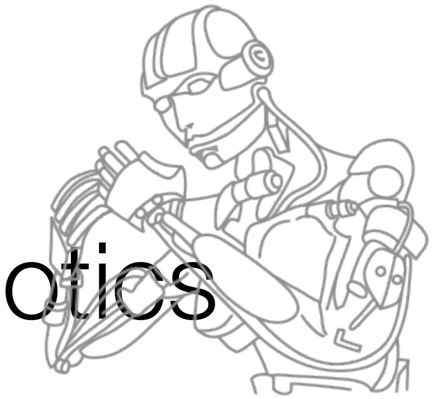
Reinforcement Learning: A Tutorial

Jan Peters

Supervised/Unsupervised Learning



Motivation for optimal decision making in robotics



Typically, **supervised learning is not enough**

- Imperfect demonstrations
- We cannot demonstrate everything!

Exploration

Reward

ML systems need **self-improvment!**

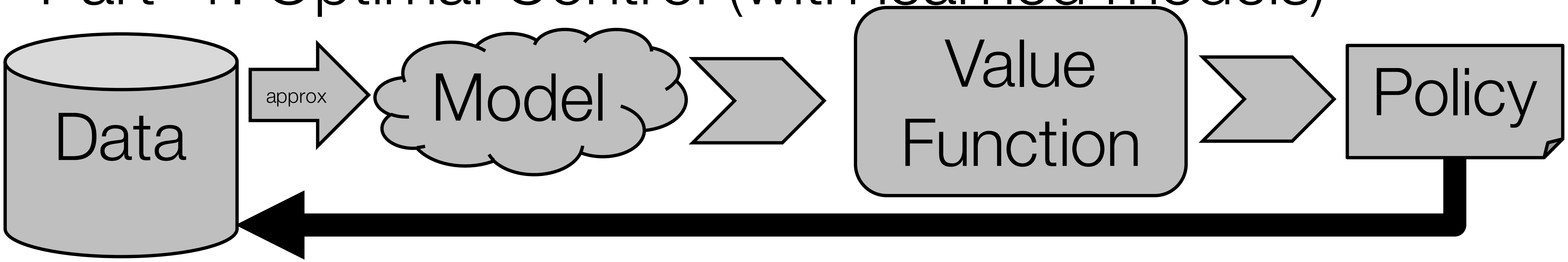
- The system explores by trial and error
- We give evaluative feedback → reward

Today, we are going to look at the problem of how to derive optimal actions that maximize long-term reward

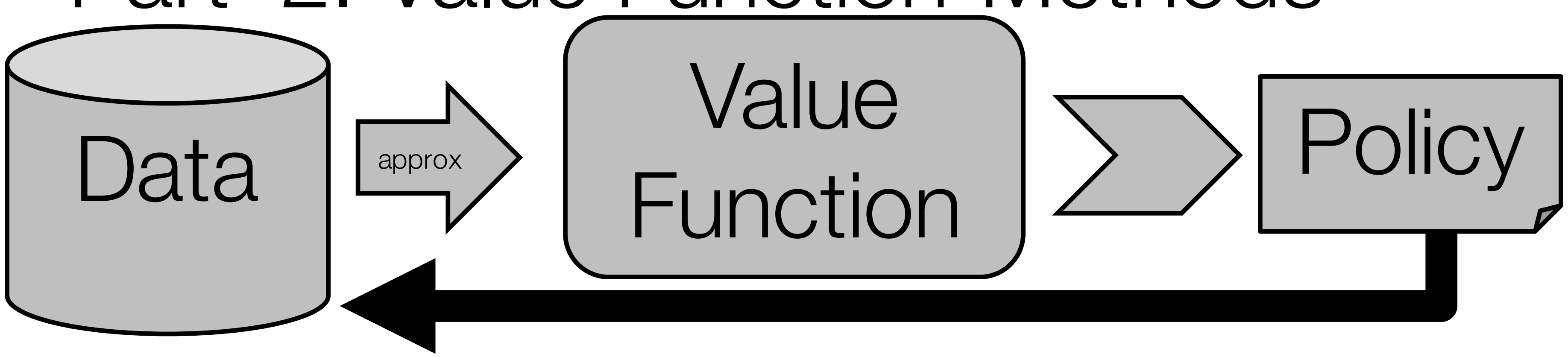
Note:
reward = - cost
Max(reward) = Min(cost)

Reinforcement Learning

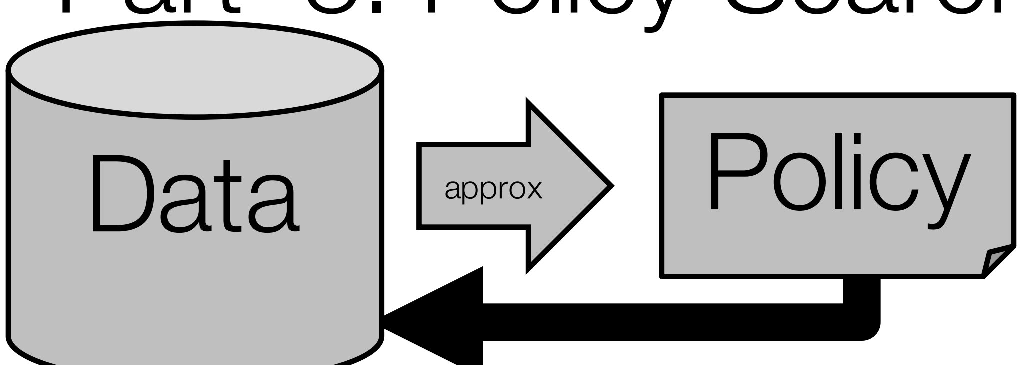
Part 1. Optimal Control (with learned models)



Part 2. Value Function Methods

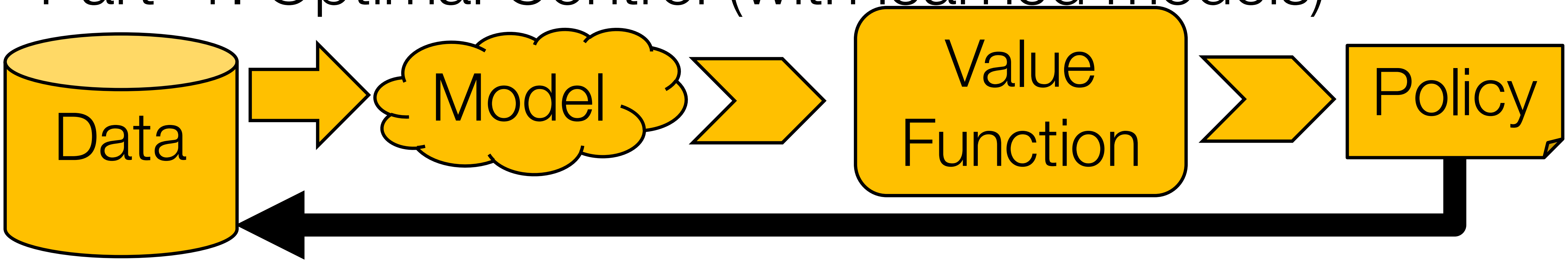


Part 3. Policy Search

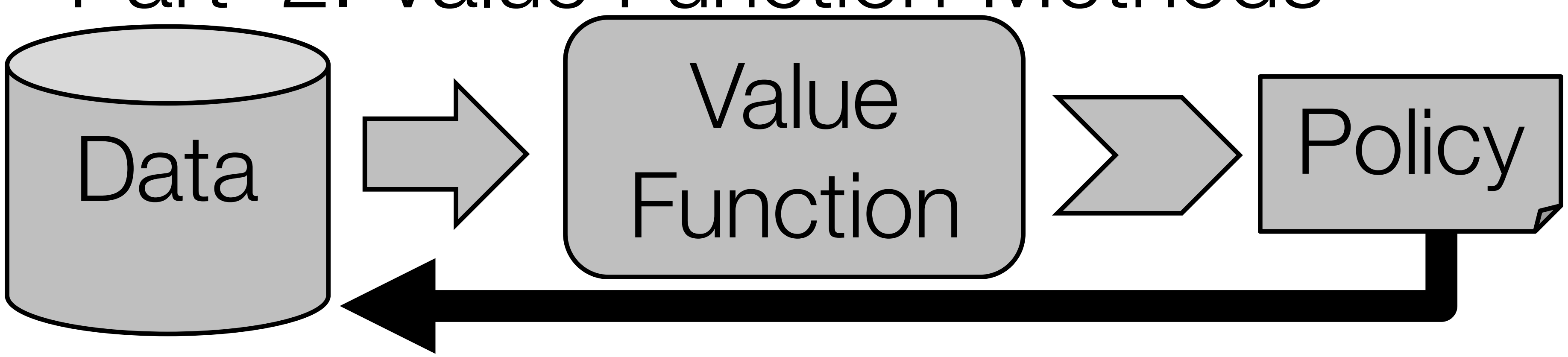


Reinforcement Learning

Part 1. Optimal Control (with learned models)



Part 2. Value Function Methods



Part 3. Policy Search

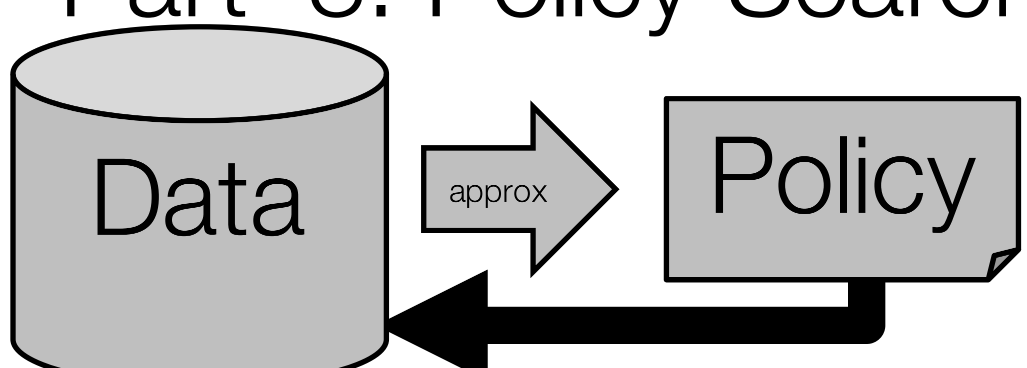
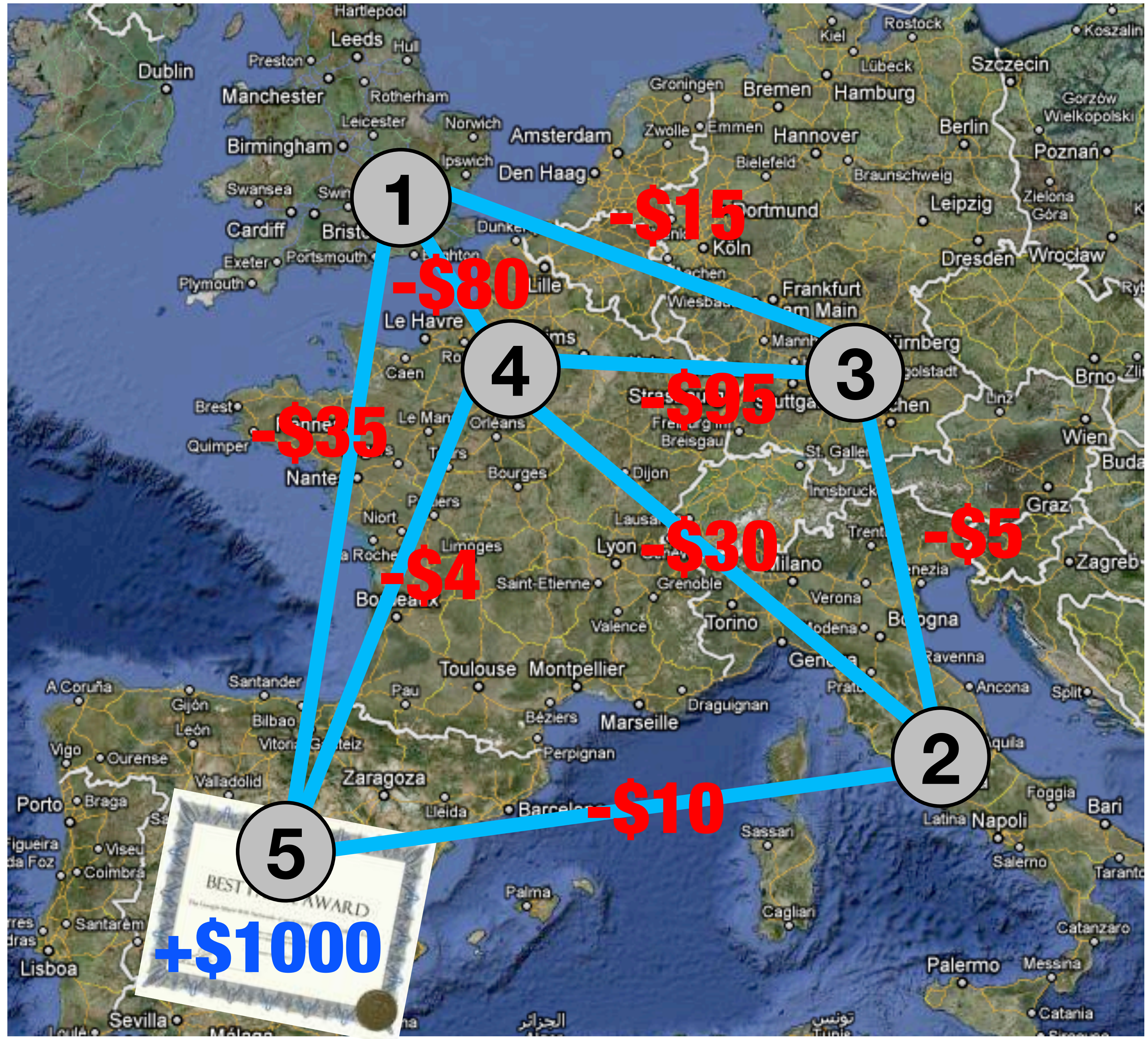


Illustration of basic idea…

You have won a Best-Paper Award in Madrid!

What is the Optimal Policy to Collect it?

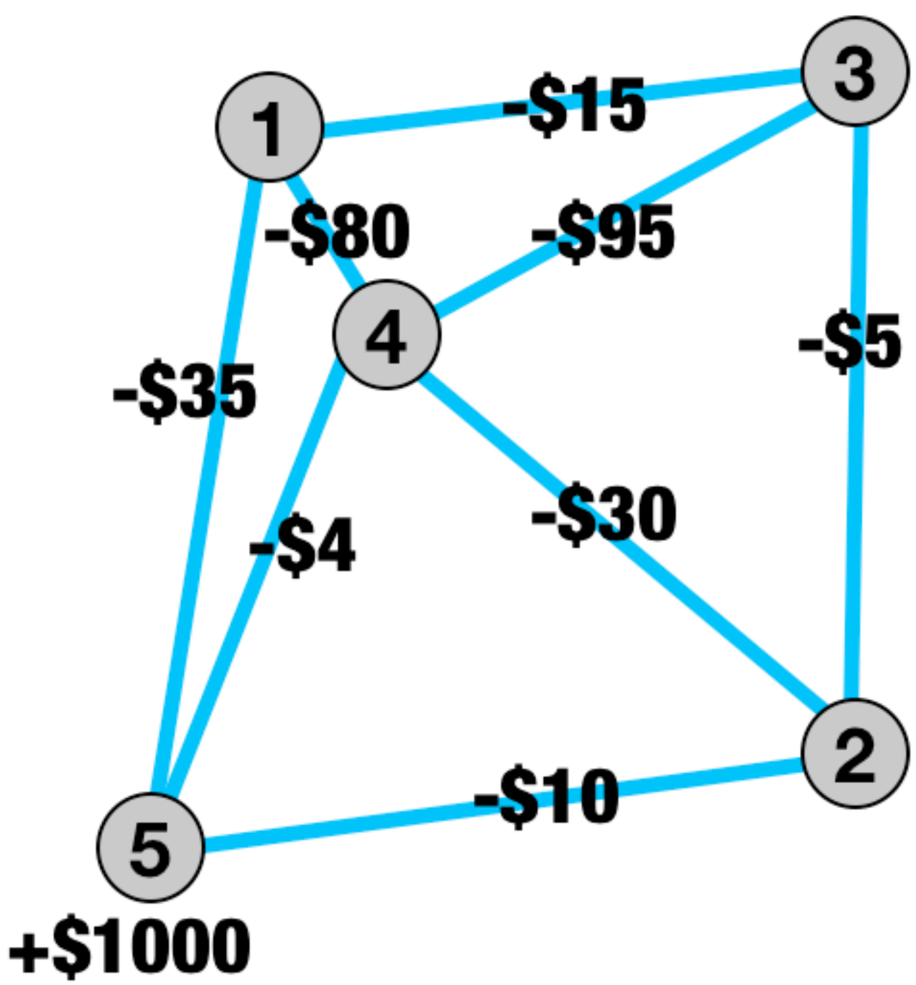


Dynamic Programming

*"An optimal sequence of controls in a multistage optimization problem has the property that **whatever the initial stage, state and controls are**, the **remaining controls** must constitute **an optimal sequence of decisions for the remaining problem** with stage and state resulting from previous controls considered as initial conditions."*

Richard Bellman, Dynamic Programming, 1957

Let’s Try this Example!

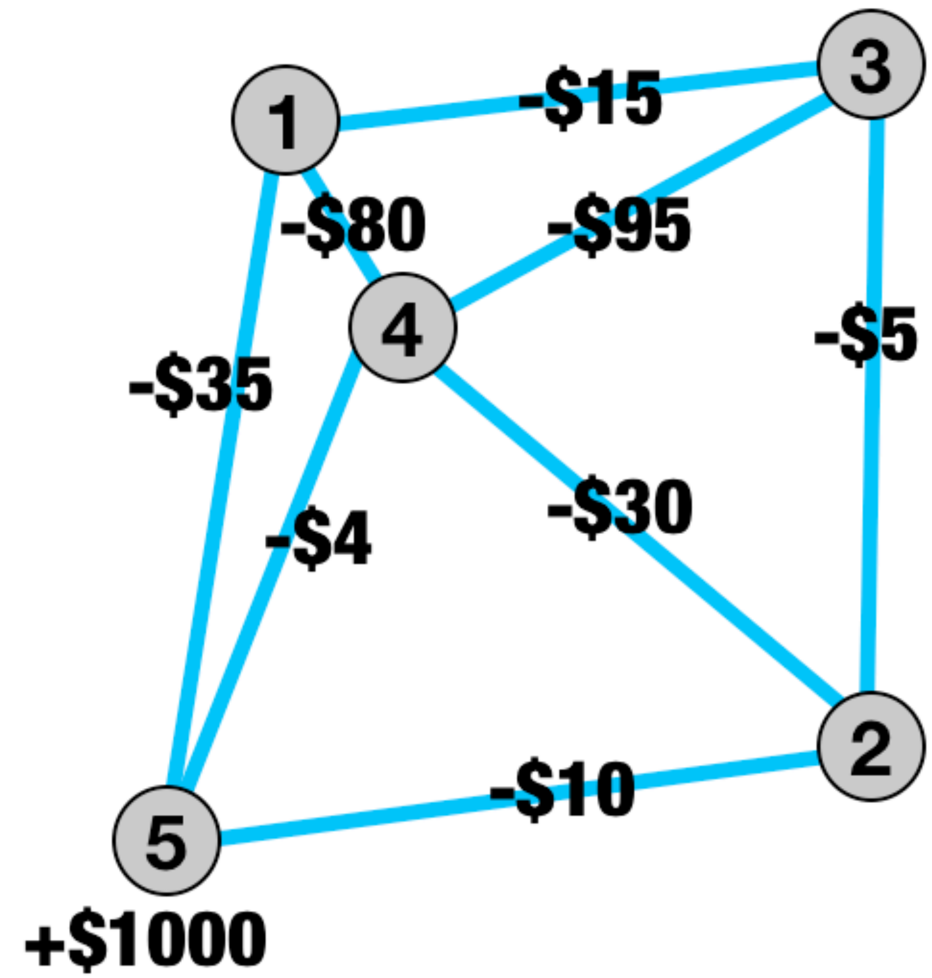


	T-4	T-3	T-2	T-1	T	
					0	1
					0	2
					0	3
					0	4
					1000	5

Markov Decision Problems (MDP)

A stationary **MDP** is defined by:

- its state space $\boldsymbol{s} \in \mathcal{S}$
- its action space $\boldsymbol{a} \in \mathcal{A}$
- its transition dynamics $\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$
- its reward function $r(\boldsymbol{s}, \boldsymbol{a})$
- and its initial state probabilities $\mu_0(\boldsymbol{s})$

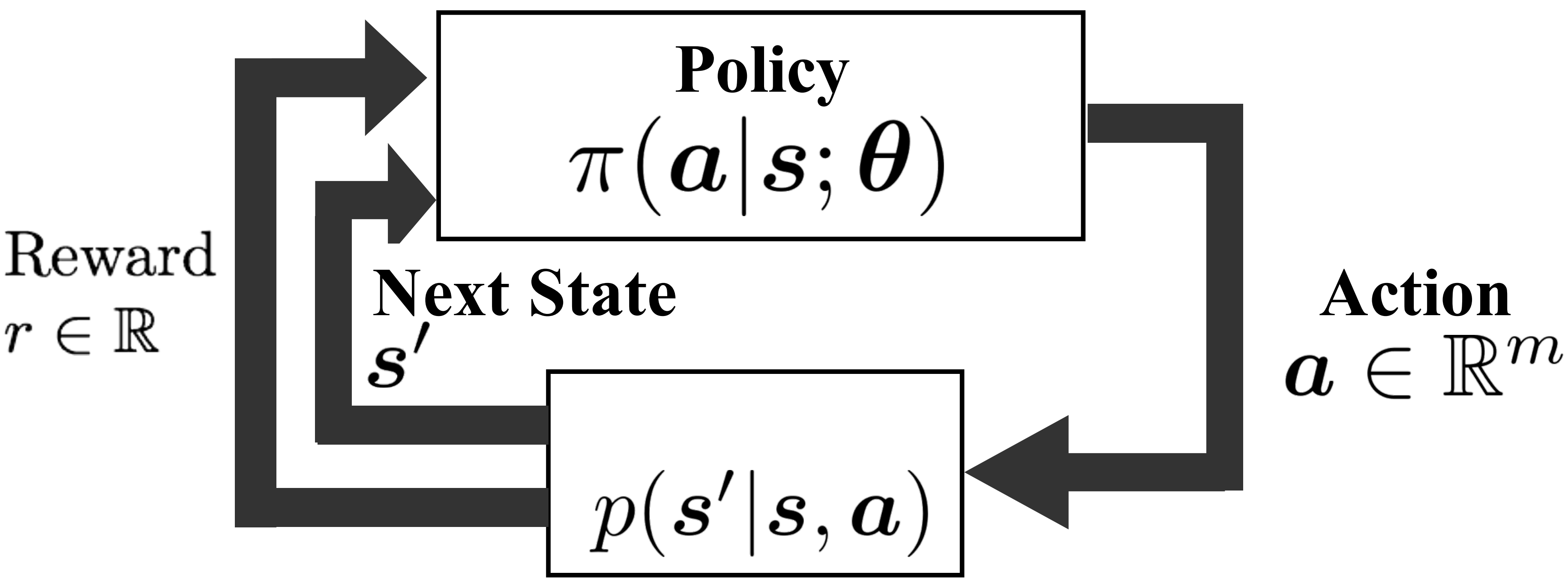


Markov property:

$$\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \ldots) = \mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$$

- Transition dynamics depends on only on the current time step

Basic Reinforcement Learning Loop:



Goal: Maximize the expected long-term reward

$$J_{\boldsymbol{\theta}} = \mathbb{E}_{\mu_0,\mathcal{P},\pi}\left[\sum_{t=1}^{T-1} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$$

discount factor $0 \leq \gamma \leq 1$

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

Algorithmic Description of Value Iteration

Init: $V_T^*(s) \leftarrow r_T(s),\ t = T$

Repeat $t = t - 1$

Compute Q-Function for time step t (for each state action pair)

$$Q_t^*(s,a) = r_t(s,a) + \gamma \sum_{s'} P_t(s'|s,a) V_{t+1}^*(s')$$

Compute V-Function for time step t (for each state)

$$V_t^*(s) = \max_a Q_t^*(s,a)$$

Until t = 1

Return: Optimal policy for each time step

$$\pi_t^*(s) = \operatorname{argmax}_a Q_t^*(s,a)$$

Howard, Value Iteration, 1958

What if you

„Bellman Equation“ (Bellman Principle of Optimality)

$$V^*(\boldsymbol{s}) = \max_{\boldsymbol{a}} \left(r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\mathcal{P}} \left[V^*(\boldsymbol{s}') \middle| \boldsymbol{s}, \boldsymbol{a} \right] \right)$$

⇨ Iterating the Bellman Equation converges to the stationary value function V^*

Alternatively, we can write this in Q-Functions

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) = r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\mathcal{P}} \left[\max_{\boldsymbol{a}'} Q^*(\boldsymbol{s}', \boldsymbol{a}') \middle| \boldsymbol{s}, \boldsymbol{a} \right]$$

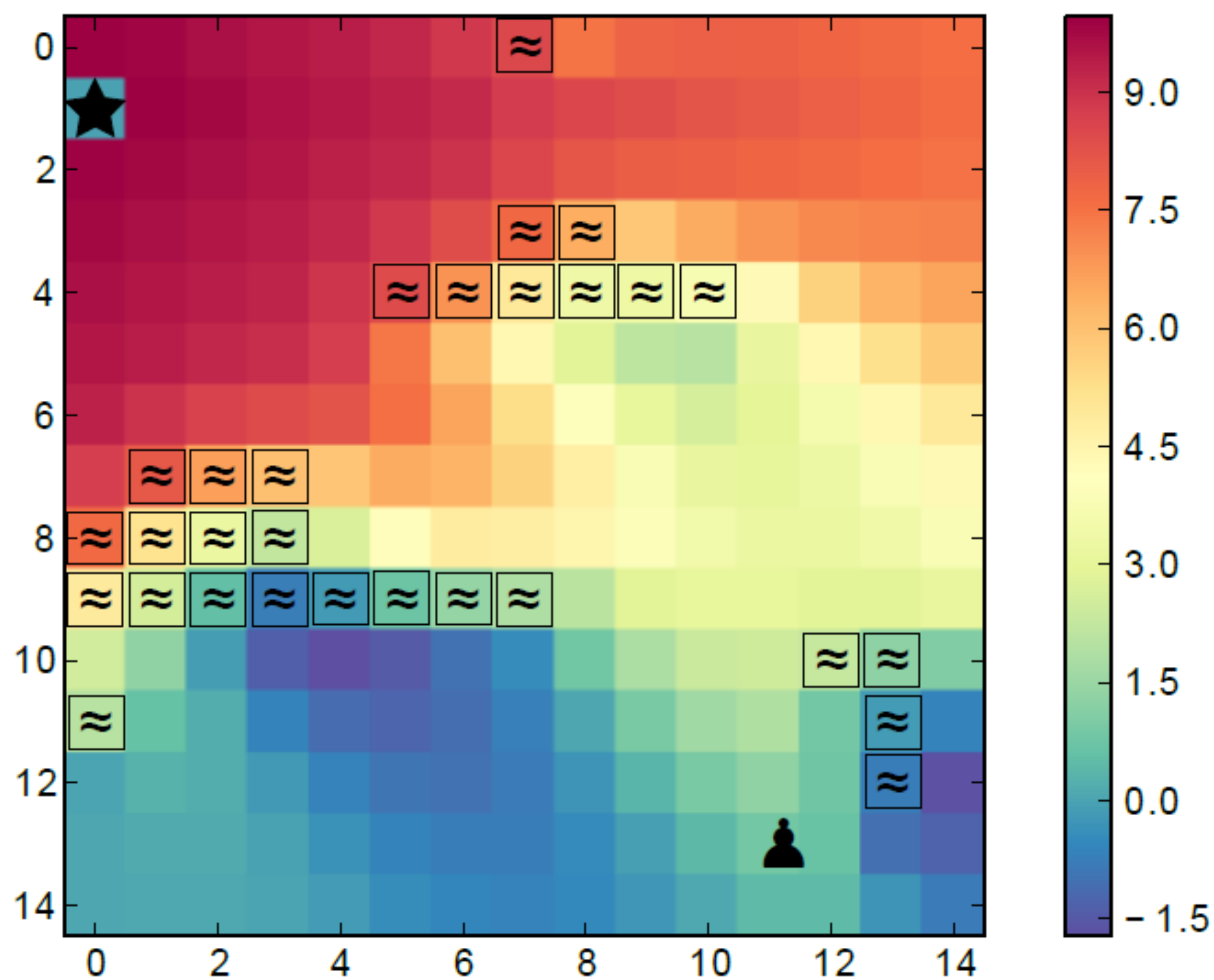
Richard Bellman, Dynamic Programming, 1957

If your life is infinite: Stationary value functions

An Illustration…

Policy always goes directly to the star

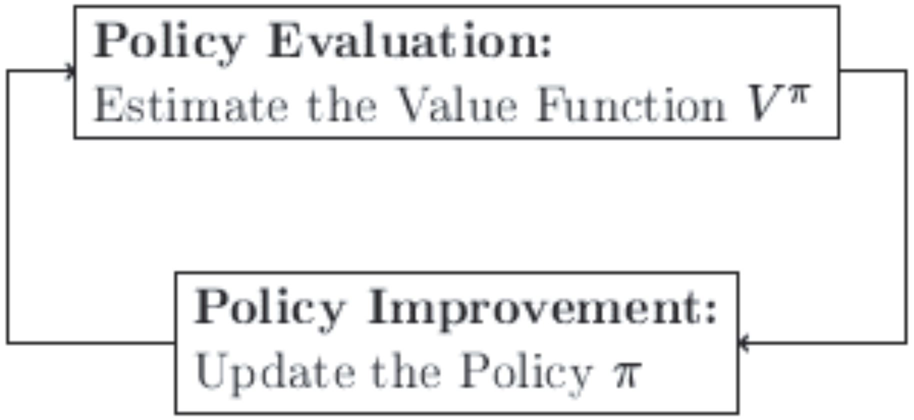
Going through puddles is punished



Dann, et al: *Policy Evaluation with Temporal Differences: A survey and comparison*, JMLR, 2014

What if the max is expensive?

Typically done iteratively:



- Policy Evaluation:

  Estimate quality of states (and actions) with current policy

- Policy Improvement:

  Improve policy by taking actions with the highest quality

Such iterations are called **Policy Iteration.**

Dann, et al: *Policy Evaluation with Temporal Differences: A survey and comparison*, JMLR, 2014

A Special MDP:
Linear Quadratic Gaussian Systems

An LQR system is defined as

- its state space $\boldsymbol{x} \in \mathbb{R}^n$ (note: same as $\boldsymbol{s}$)
- its action space $\boldsymbol{u} \in \mathbb{R}^m$ (note: same as $\boldsymbol{u}$)
- its (possibly time-dependent) linear transition dynamics with Gaussian noise

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \boldsymbol{\Sigma}_t)$$

- its quadratic reward function

$$r_t(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x} - \boldsymbol{r}_t)^T \boldsymbol{R}_t (\boldsymbol{x} - \boldsymbol{r}_t) + \boldsymbol{u}_t^T \boldsymbol{H}_t \boldsymbol{u}_t$$

$$r_T(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{r}_T)^T \boldsymbol{R}_T (\boldsymbol{x} - \boldsymbol{r}_T)$$

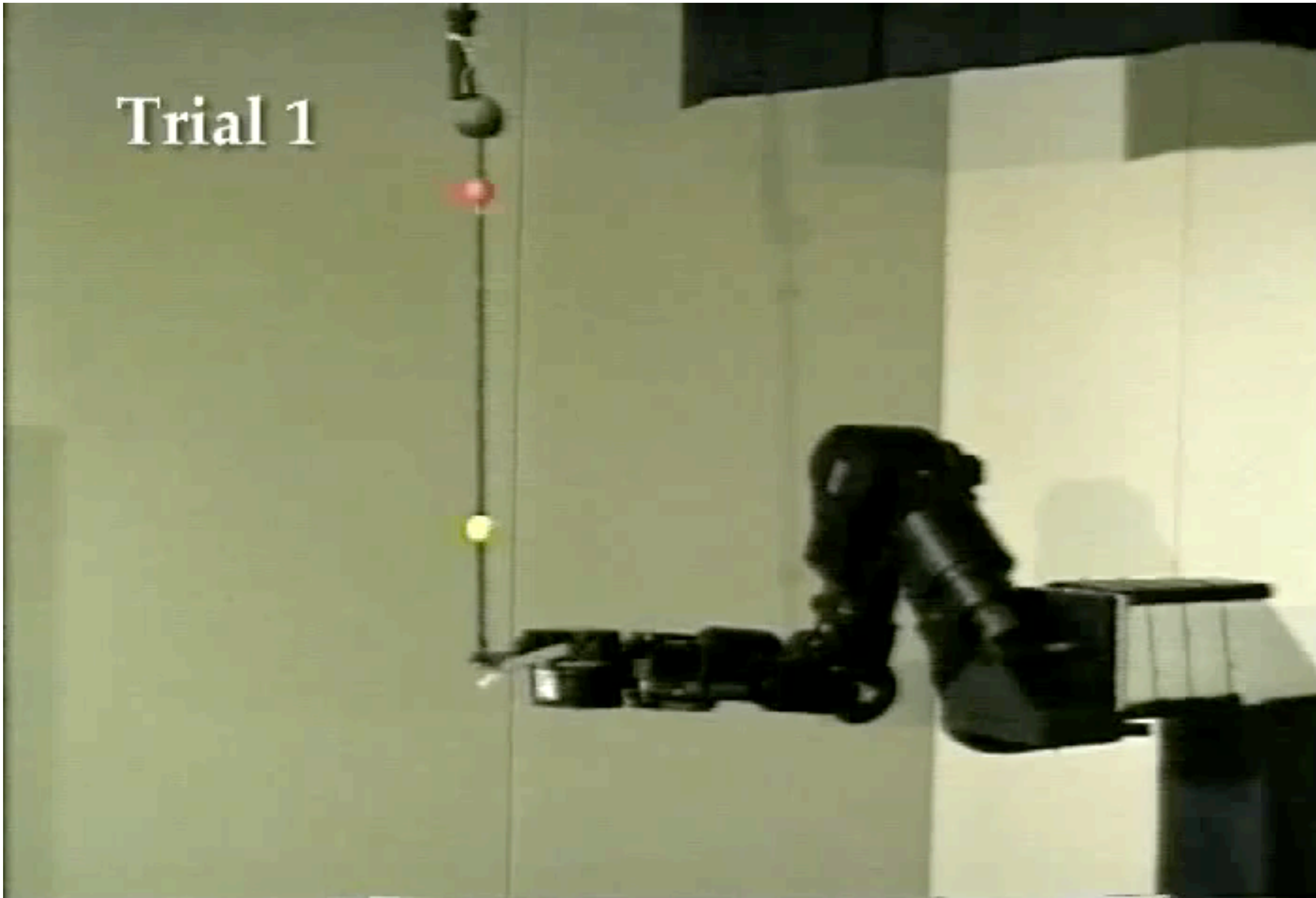
- and its initial state density

$$\mu_0(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$$

Rudollf Emil Kalman, Optimal Control, 1959

Stefan Schaal & Christopher G. Atkeson, 1998

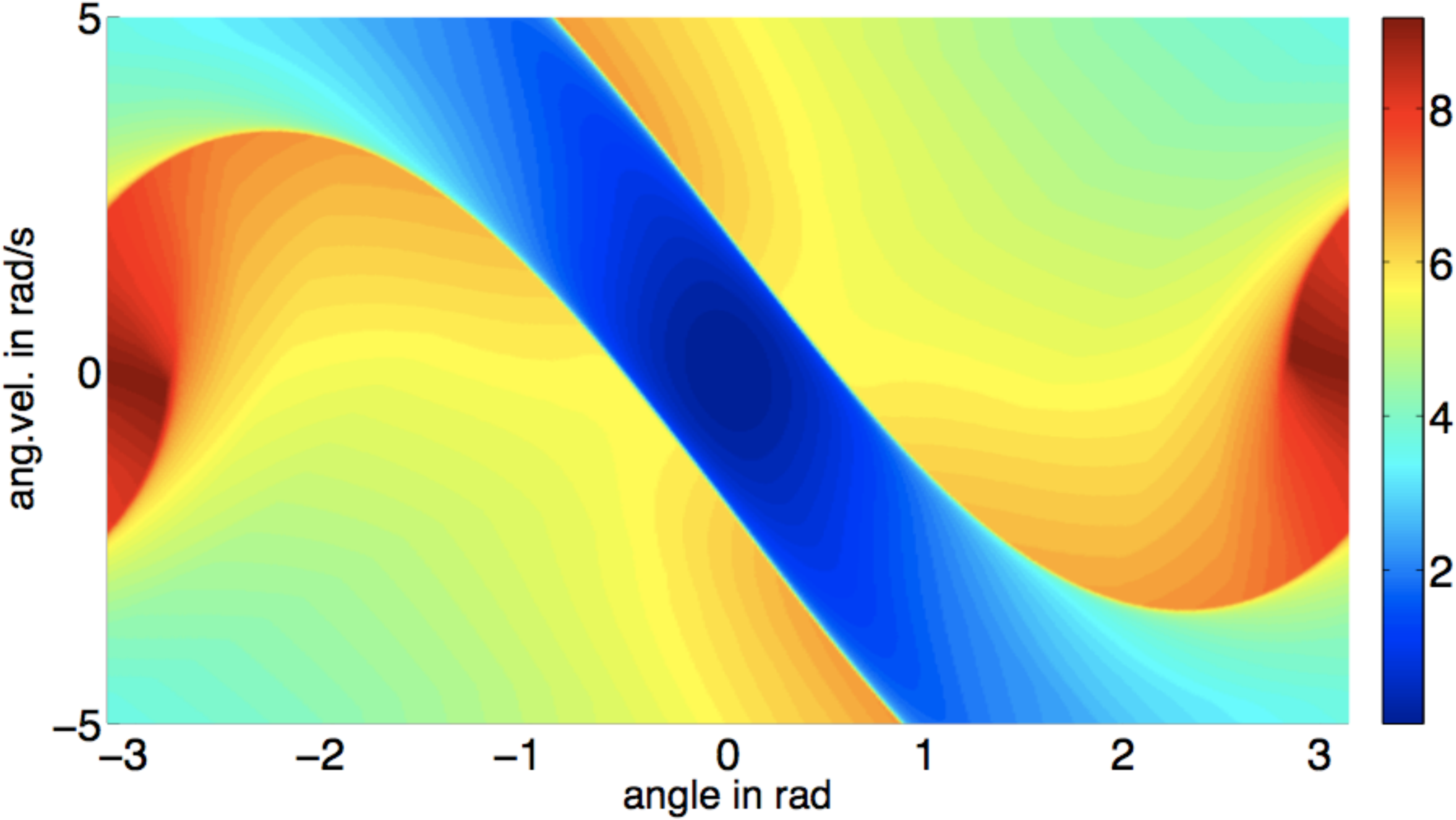
A Special MDP: Linear Quadratic Gaussian Systems



Deisenroth et al., „Gaussian Process Dynamic Programming“, Neurocomputing 2009

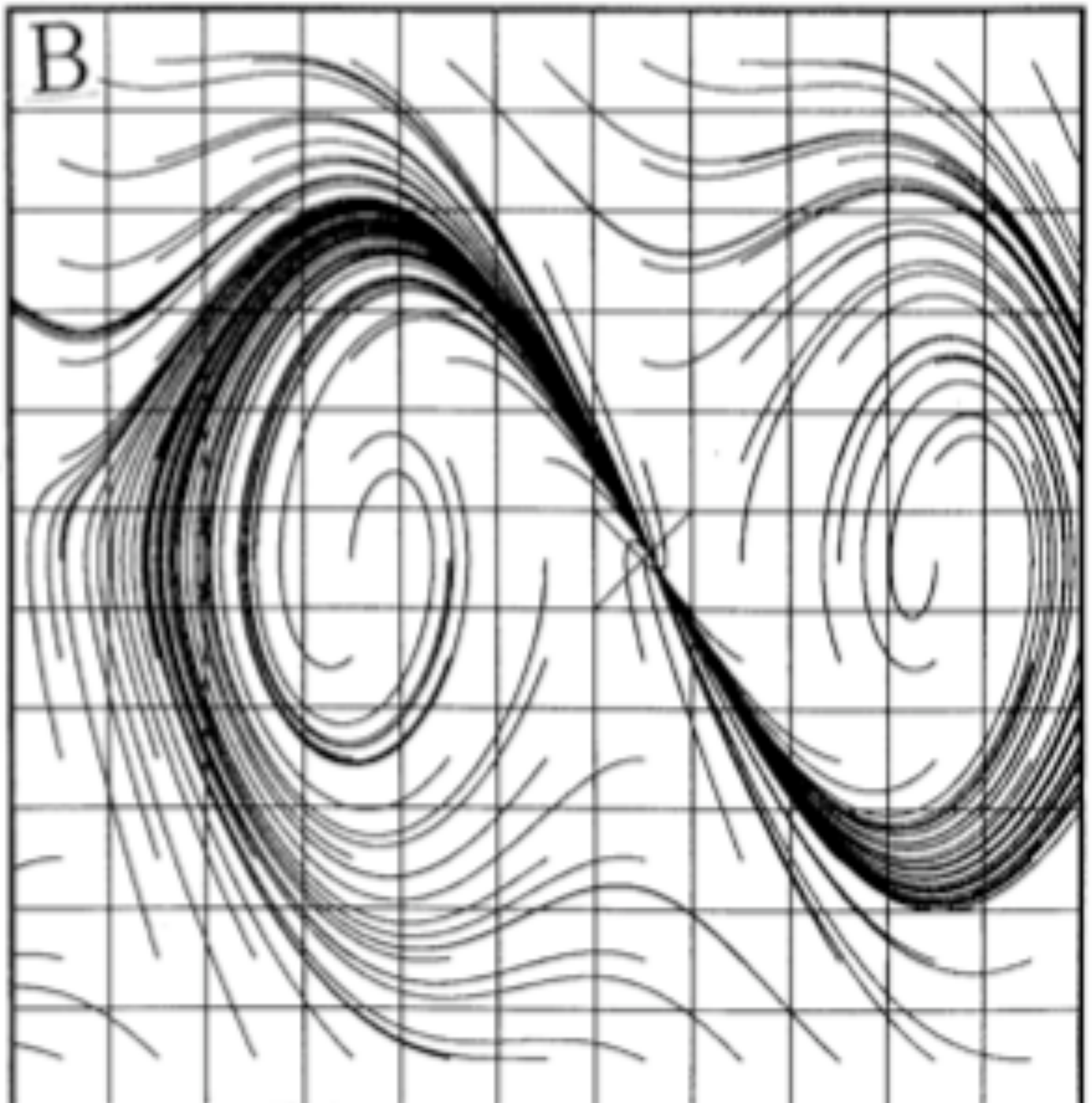
What’s wrong with LQR?

Value function for the inverted pendulum (on costs = negative rewards)



Highly non-linear function (certainly not quadratic)

Possible: Learn Solutions only where needed!



If you know places where we start...

... we can just look ahead and approximate the solution locally around an initial trajectory

Christopher G. Atkeson, 1996

Local Solutions by Linearizations

Every smooth function can be modeled with a Taylor expansion

$$f(\mathbf{x}) = f(\mathbf{a}) + \frac{df}{d\mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{a}} (\mathbf{x}-\mathbf{a}) + \frac{1}{2}(\mathbf{x}-\mathbf{a})^T \frac{d^2 f}{d\mathbf{x}^2}\bigg|_{\mathbf{x}=\mathbf{a}} (\mathbf{x}-\mathbf{a}) + \ldots$$

Hence, we can also **approximate the (learned) forward dynamics by linearizing** at the point $(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{u}}_t)$

$$\boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx f(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{u}}_t) + \frac{df}{d\boldsymbol{s}}(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t) + \frac{df}{d\boldsymbol{u}}(\boldsymbol{u}_t - \tilde{\boldsymbol{u}}_t)$$

$$= \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t + \boldsymbol{b}_t$$

and **approximate the (learned) reward function by a second order approximation**

$$r_t(\boldsymbol{s}_t, \boldsymbol{a}_t) \approx r(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{u}}_t) + \frac{dr}{d\boldsymbol{x}}(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t) + (\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t)^T \frac{dr}{d\boldsymbol{x} d\boldsymbol{x}}(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t) - \boldsymbol{u}_t^T \boldsymbol{H}_t \boldsymbol{u}_t$$

Dyer & McReynolds, Jacobson & Maine, 1969

Local Solutions by Linearizations

So we are back to the **full linear optimal control case with**…

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \boldsymbol{\Sigma}_t)$$

$$r(\boldsymbol{x}, \boldsymbol{u}) = -\boldsymbol{x}^T\boldsymbol{R}_t\boldsymbol{x} + 2\boldsymbol{r}_t^T\boldsymbol{x} - \boldsymbol{u}^T\boldsymbol{H}_t\boldsymbol{u} + \text{const}$$

that we know how to solve…

Hence our algorithm for **solving non-linear optimal control** is…

1. **Backward Solution:** Compute optimal control law (i.e. Gains $\boldsymbol{K}_t$ and offsets $\boldsymbol{k}_t$

2. **Forward Propagation:** Run simulator with optimal control law to obtain linearization points $(\tilde{\boldsymbol{x}}_{1:T}, \tilde{\boldsymbol{u}}_{1:T})$

1.If not converged, go to 1.

Dyer & McReynolds, Jacobson & Maine, 1969

Christopher G. Atkeson & Stefan Schaal, 1997

Application to the Swing-Up

Some interesting results (only in simulation)

Work by Emo Todorov and Yuval Tassa
(They call basically the same algorithm incremential LQG, iLQG).

Note: iLQG is just a simplification of Differential Dynamic Programming (Dyer & McReynolds, 1969)

Synthesis of Complex Behaviors
with
Online Trajectory Optimization

(under review)

Wrap-Up: Optimal Control

We now know how to compute optimal policies

Cool, thats all we need. Lets go home…

Wait, there is a catch! Unfortunately, we can only do this in 2.5 cases

- Discrete Systems

  Easy: integrals turn into sums

  …but the world is not discrete!

- Linear Systems, Quadratic Reward, Gaussian Noise (LQR)

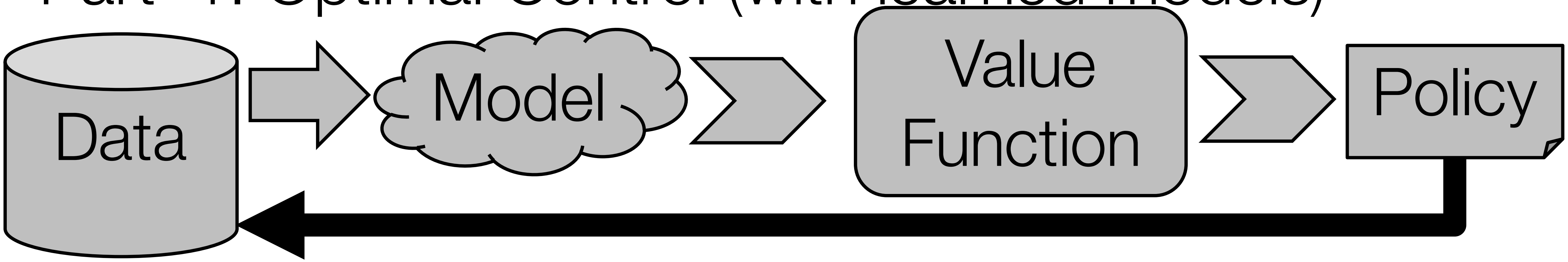
  … but the world is not linear!

- Along an optimal trajectory – finding it is really hard!

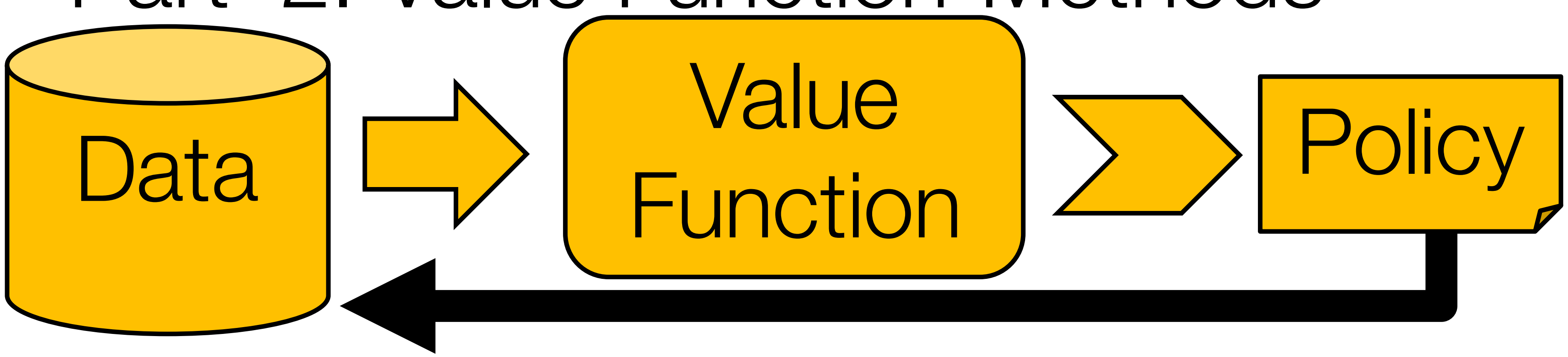
Otherwise, we need to approximate!

Reinforcement Learning

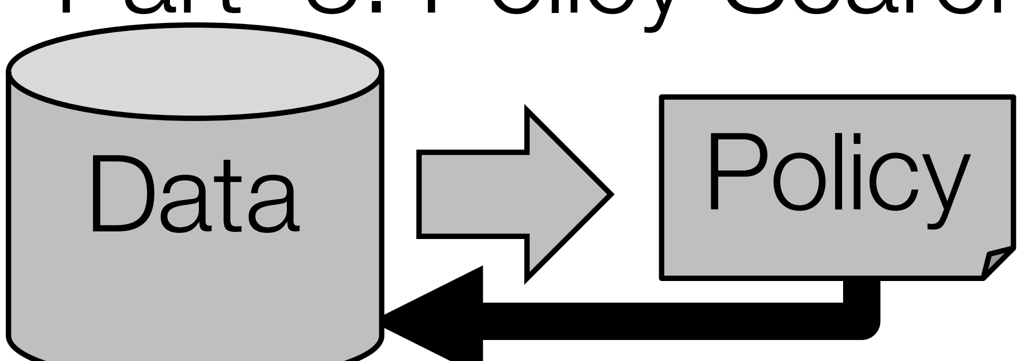
Part 1. Optimal Control (with learned models)



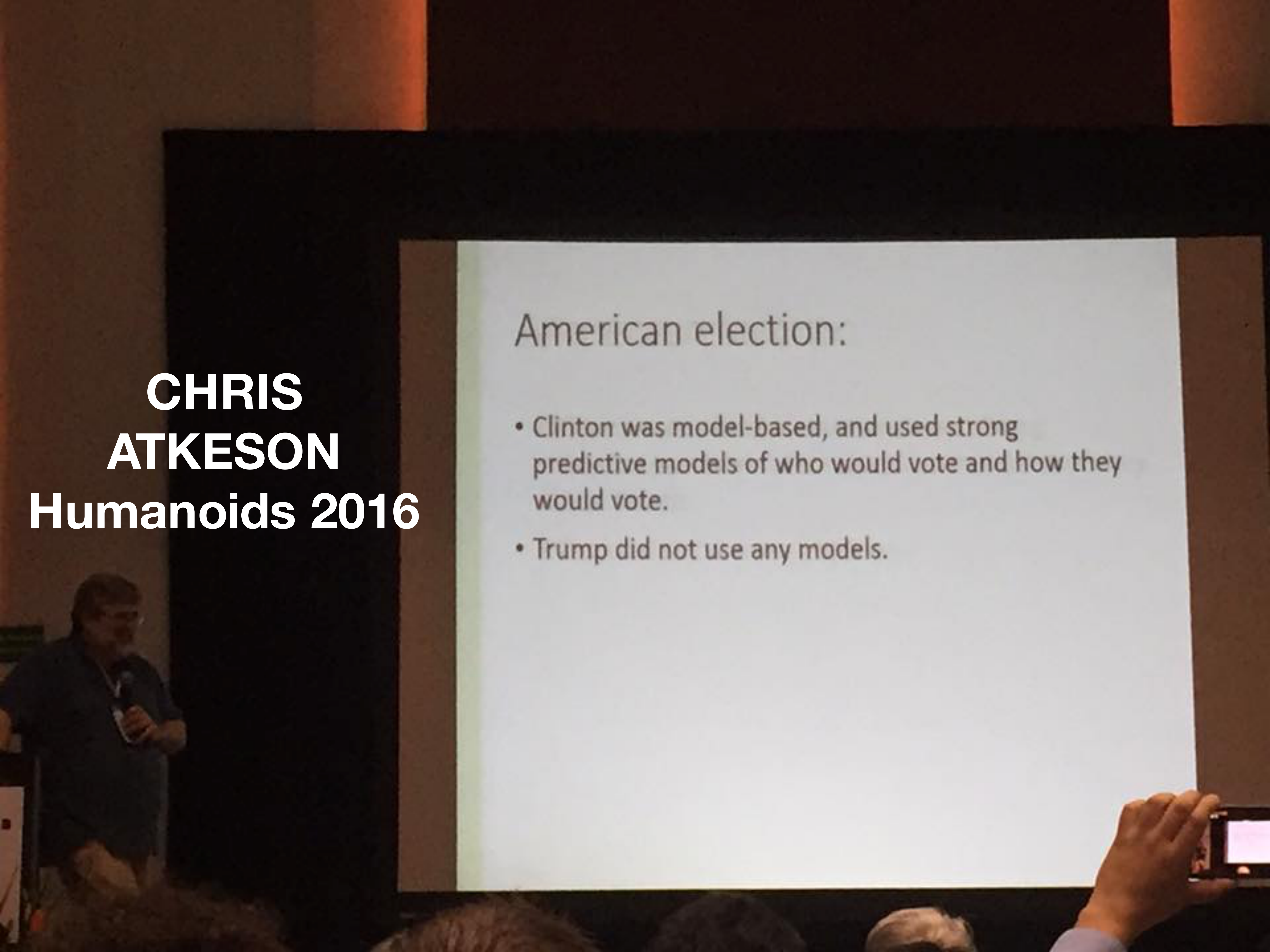
Part 2. Value Function Methods



Part 3. Policy Search



CHRIS
ATKESON
Humanoids 2016



Purpose of this Lecture

Often, learning a good model is too hard

➡The optimization inherent in optimal control is prone to model errors, as the controller may achieve the objective only because model errors get exploited

➡Optimal control methods based on linearization of the dynamics work only for moderately non-linear tasks

➡(Ideally model-free) Approaches are needed that do not make any assumption on the structure of the model

Classical Reinforcement Learning:

➡Solve the optimal control problem by learning the value function, not the model!

Markov Decision Processes (MDP)

Classical reinforcement learning is typically formulated for the infinite horizon objective

Infinite Horizon: maximize **discounted accumulated reward**

$$J_{\boldsymbol{\pi}} = \mathbb{E}_{\mu_0, \mathcal{P}, \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]$$

$0 \leq \gamma < 1$ … discount factor (note change!)

Trades-off long term vs. immediate reward

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

Value functions of a policy

Value function and state-action value function of a policy can be computed iteratively

$$
\begin{aligned}
V^{\pi}(\boldsymbol{s}) &= \mathbb{E}_{\pi}\Big[r(\boldsymbol{s},\boldsymbol{a}) + \gamma \mathbb{E}_{\mathcal{P}}\left[V^{\pi}(\boldsymbol{s}')\right] \Big| \boldsymbol{s}\Big] \\
&= \int \pi(\boldsymbol{a}|\boldsymbol{s})\Big(r(\boldsymbol{s},\boldsymbol{a}) + \gamma \int \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) V^{\pi}(\boldsymbol{s}') d\boldsymbol{s}'\Big) d\boldsymbol{a}
\end{aligned}
$$

$$
\begin{aligned}
Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) &= r(\boldsymbol{s},\boldsymbol{a}) + \gamma \mathbb{E}_{\mathcal{P},\pi}\Big[Q^{\pi}(\boldsymbol{s}',\boldsymbol{a}') \Big| \boldsymbol{s},\boldsymbol{a}\Big] \\
&= r(\boldsymbol{s},\boldsymbol{a}) + \gamma \int \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) \int \pi(\boldsymbol{a}'|\boldsymbol{s}') Q^{\pi}(\boldsymbol{s}',\boldsymbol{a}') d\boldsymbol{a}' d\boldsymbol{s}'
\end{aligned}
$$

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

Value-based Reinforcement Learning

Classical Reinforcement Learning

Updates the value function based on samples

$$\mathcal{D} = \{\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i\}_{i=1\ldots N}$$

We do not have a model and we do not want to learn it

Use the samples to update Q-function (or V-function)

Lets start simple:

Discrete states/actions → Tabular Q-function

Temporal difference learning

Given a transition (s_t, a_t, r_t, s_{t+1})**, we want to update the V-function**

- Use the estimate of the current value: $V(s_t)$
- 1-step prediction of the current value: $\hat{V}(s_t) = r_t + \gamma V(s_{t+1})$
- 1-step prediction error (called temporal difference (TD) error)

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Update current value with the temporal difference error

$$V_{\text{new}}(s_t) = V(s_t) + \alpha\delta_t = (1-\alpha)V(s_t) + \alpha(r_t + \gamma V(s_{t+1}))$$

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

Temporal difference learning

The **TD error**

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

compares the **one-time step lookahead prediction**

$$\hat{V}(s_t) = r_t + \gamma V(s_{t+1})$$

with the **current estimate** of the value function $V(s_t)$

- if $\hat{V}(s_t) > V(s_t)$ than $V(s_t)$ is increased
- if $\hat{V}(s_t) < V(s_t)$ than $V(s_t)$is decreased

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

Algorithmic Description of TD Learning

Init: $V_0^*(s) \leftarrow 0$

Repeat $t = t + 1$

Observe transition (s_t, a_t, r_t, s_{t+1})

Compute TD error $\delta_t = r_t + \gamma V_t(s_{t+1}) - V_t(s_t)$

Update V-Function $V_{t+1}(s_t) = V_t(s_t) + \alpha \delta_t$

until convergence of V

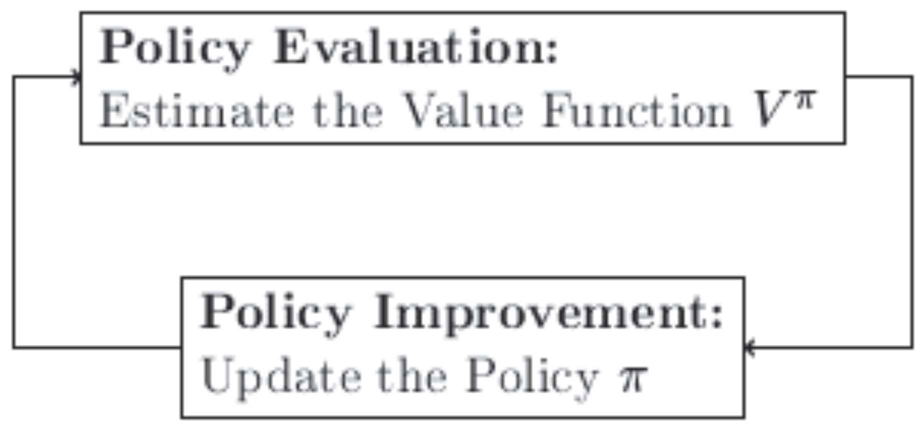
➡ Used to compute Value function of behavior policy

➡ Sample-based version of policy evaluation

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

Temporal difference learning for control

So far: Policy evaluation with TD methods



Can **we also do the policy improvement step with samples**?

Yes, but we need to enforce exploration!

Epsilon-Greedy Policy: $$\pi(\boldsymbol{a}|\boldsymbol{s}) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}|, & \text{if } \boldsymbol{a} = \text{argmax}_{\boldsymbol{a}'} Q^\pi(\boldsymbol{s}, \boldsymbol{a}') \\ \epsilon/|\mathcal{A}|, & \text{otherwise} \end{cases}$$

Soft-Max Policy: $$\pi(\boldsymbol{a}|\boldsymbol{s}) = \frac{\exp(\beta Q(\boldsymbol{s}, \boldsymbol{a}))}{\sum_{\boldsymbol{a}'} \exp(\beta Q(\boldsymbol{s}, \boldsymbol{a}'))}$$

➡ **Do not always take greedy action**

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

Temporal difference learning for control

Update equations for learning the Q-function $Q(s, a)$

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\delta_t, \quad \delta_t = r_t + \gamma Q_t(s_{t+1}, a_?) - Q_t(s_t, a_t)$$

Two different methods to estimate $a_?$

Q-learning: $a_? = \text{argmax}_a Q_t(s_{t+1}, a)$

Estimates Q-function of optimal policy

Off-policy samples: $a_? \neq a_{t+1}$

SARSA: $a_? = a_{t+1}$ **,** where $a_{t+1} \sim \pi(a|s_{t+1})$

Estimates Q-function of exploration policy

On-policy samples

Note: The policy for generating the actions depends on the Q-function $\Rightarrow$ non-stationary policy

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

Approximating the Value Function

In the continuous case, we need to approximate the V-function (except for LQR)

Lets keep it simple, we use **a linear model** to represent the V-function

$$V^{\pi}(\boldsymbol{s}) \approx V_{\boldsymbol{\omega}}(\boldsymbol{s}) = \boldsymbol{\phi}^T(\boldsymbol{s})\boldsymbol{\omega}$$

How can we find the parameters ω ?

➡ Again with Temporal Difference Learning

Rich Sutton & Andy Barto, Reinforcement Learning, 1998

TD-learning with Function Approximation

Derivation:

Use the **recursive definition of V-function**:

$$\mathrm{MSE}(\boldsymbol{\omega}) \approx \mathrm{MSE}_{\mathrm{BS}}(\boldsymbol{\omega}) = 1/N \sum_{i=1}^{N} \left(\hat{V}^{\pi}(\boldsymbol{s}_i) - V_{\boldsymbol{\omega}}(\boldsymbol{s}_i) \right)^2$$

with $\hat{V}^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}\left[r(\boldsymbol{s}, \boldsymbol{a}) + \mathbb{E}_{\mathcal{P}}\left[V_{\boldsymbol{\omega}_{\mathrm{old}}}(\boldsymbol{s}') | \boldsymbol{s}, \boldsymbol{a} \right] \right]$

⇨ **Bootstrapping (BS):** Use the old approximation to get the target values for a new approximation

How can we **minimize** this function ?

Lets use **stochastic gradient descent**

Dann, et al: *Policy Evaluation with Temporal Differences: A survey and comparison*, JMLR, 2014

Temporal difference learning

Stochastic gradient descent on our error function *MSE*$_{BS}$

$$\mathrm{MSE}_{\mathrm{BS,t}}(\boldsymbol{\omega}) = 1/N \sum_{i=1}^{N} \left(\hat{V}(\boldsymbol{s}_t) - V_{\boldsymbol{\omega}}(\boldsymbol{s}_i)\right)^2$$

$$= 1/N \sum_{i=1}^{N} \left(r_i + \gamma V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_i') - V_{\boldsymbol{\omega}}(\boldsymbol{s}_i)\right)^2$$

Update rule (for current time step t, $V_{\boldsymbol{\omega}}(\boldsymbol{s}) = \boldsymbol{\phi}^T(\boldsymbol{s})\boldsymbol{\omega}$)

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_t \left.\frac{d\mathrm{MSE}_{\mathrm{BS}}}{d\boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}_t}$$

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha\Big(r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_{t+1}) - V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_t)\Big)\boldsymbol{\phi}^T(\boldsymbol{s}_t)$$

$$= \boldsymbol{\omega}_t + \alpha\delta_t\boldsymbol{\phi}^T(\boldsymbol{s}_t)$$

with $\delta_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_{t+1}) - V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_t)$

Dann, et al: *Policy Evaluation with Temporal Differences: A survey and comparison*, JMLR, 2014

Temporal difference learning

TD with function approximation

$$\boldsymbol{\omega}_t = \boldsymbol{\omega}_t + \alpha \delta_t \boldsymbol{\phi}^T(\boldsymbol{s}_t)$$

Difference to discrete algorithm:

- TD-error is correlated with the feature vector
- Equivalent if tabular feature coding is used, i.e., $\phi(s_i) = \boldsymbol{e}_i$

Similar update rules can be obtained for SARSA and Q-learning

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha \Big(r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma Q_{\boldsymbol{\omega}_t}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_?) - Q_{\boldsymbol{\omega}_t}(\boldsymbol{s}_t, \boldsymbol{a}_t) \Big) \boldsymbol{\phi}^T(\boldsymbol{s}_t, \boldsymbol{a}_t)$$

where $Q_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}) \approx \boldsymbol{\phi}^T(\boldsymbol{s}, \boldsymbol{a}) \boldsymbol{\omega}$

Dann, et al: *Policy Evaluation with Temporal Differences: A survey and comparison*, JMLR, 2014

Temporal difference learning

Some remarks on temporal difference learning:

- Its **not a proper** stochastic gradient descent!!
- **Why?** Target values $\hat{V}^{\pi}(\boldsymbol{s})$ **change after each parameter update!**

  We ignore the fact that $\hat{V}^{\pi}(\boldsymbol{s})$ also depends on $\boldsymbol{\omega}$
- **Side note:** This „ignorance“ actually introduces a bias in our optimization, such that we are optimizing a different objective than the *MSE*
- In certain cases, we also get **divergence** (e.g. off-policy samples)
- TD-learning is very fast in terms of computation time O(#features), but not data-efficient ⇨ **each sample is just used once!**

Dann, et al: *Policy Evaluation with Temporal Differences: A survey and comparison*, JMLR, 2014

Sucessful examples

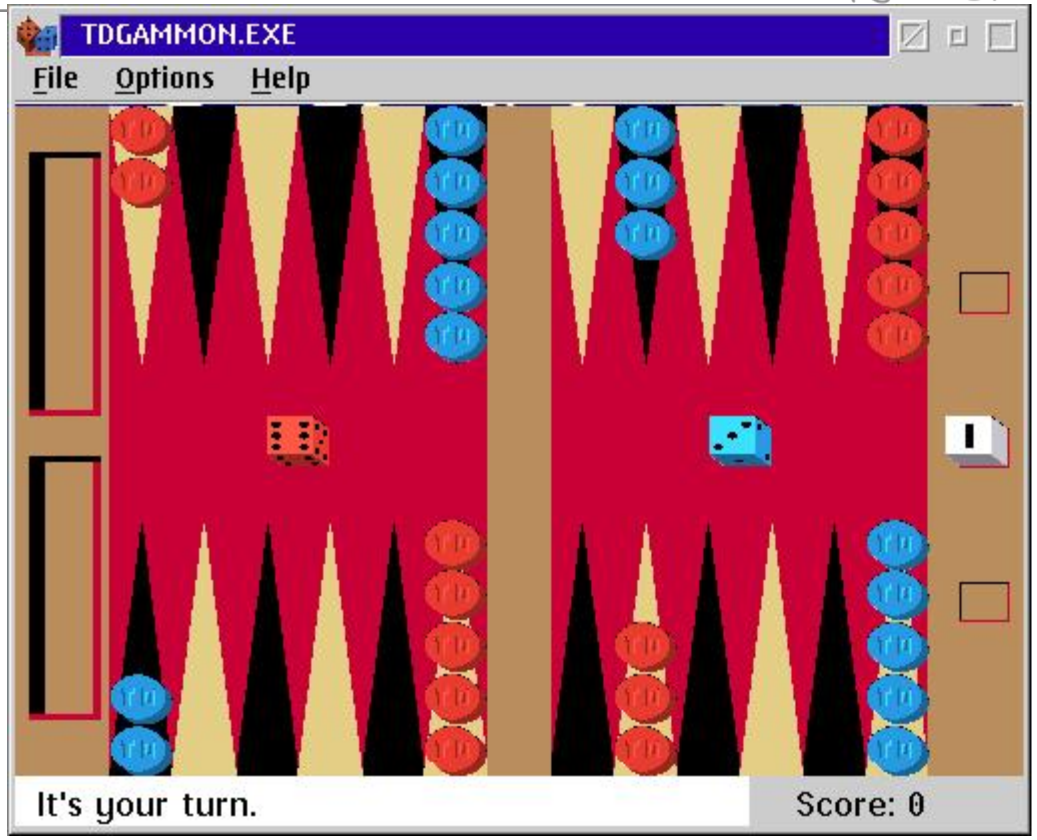
Linear function approximation

Tetris, Go

Non-linear function approximation

TD Gammon (Worldchampion level)

Atari Games (learning from raw pixel input)



Batch-Mode Reinforcement Learning

Online methods are typically **data-inefficient** as they use each data point only once

$$D = \left\{ \boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i \right\}_{i=1\ldots N}$$

Can we **re-use the whole „batch“** of data to **increase data-efficiency?**

- **Least-Squares Temporal Difference (LSTD) Learning**
- **Fitted Q-Iteration**

⇨ Computationally **much more expensive** then TD-learning!

Fitted Q-iteration

In Batch-Mode RL it is also much easier to use **non-linear function approximators**

- Many of them **only exists in the batch setup**, e.g. regression trees
- **No catastrophic forgetting**, e.g., for neural networks.
- Strong divergence problems, fixed for Neural Networks by ensuring that there is a goal state where the Q-Function value is always zero (see Lange et al. below).

Fitted Q-iteration uses non-linear function approximators for **approximate value iteration.**

Ernst, Geurts and Wehenkel, *Tree-Based Batch Mode Reinforcement Learning, JMLR 2005*
Lange, Gabel and Riedmiller. *Batch Reinforcement Learning*, *Reinforcement Learning: State of the Art*

Fitted Q-iteration

Given: Dataset $D = \left\{ \boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i \right\}_{i=1 \dots N}$

Algorithm:

Initialize $Q^{[0]}(\boldsymbol{s}, \boldsymbol{a}) = 0$, input data: $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{s}_1^T & \boldsymbol{a}_1^T \\ \vdots & \\ \boldsymbol{s}_N^T & \boldsymbol{a}_N^T \end{bmatrix}$

for k = 1 to L

Generate target values: $\tilde{q}_i^{[k]} = r_i + \gamma \max_{\boldsymbol{a}'} Q^{[k-1]}(\boldsymbol{s}'_i, \boldsymbol{a}')$

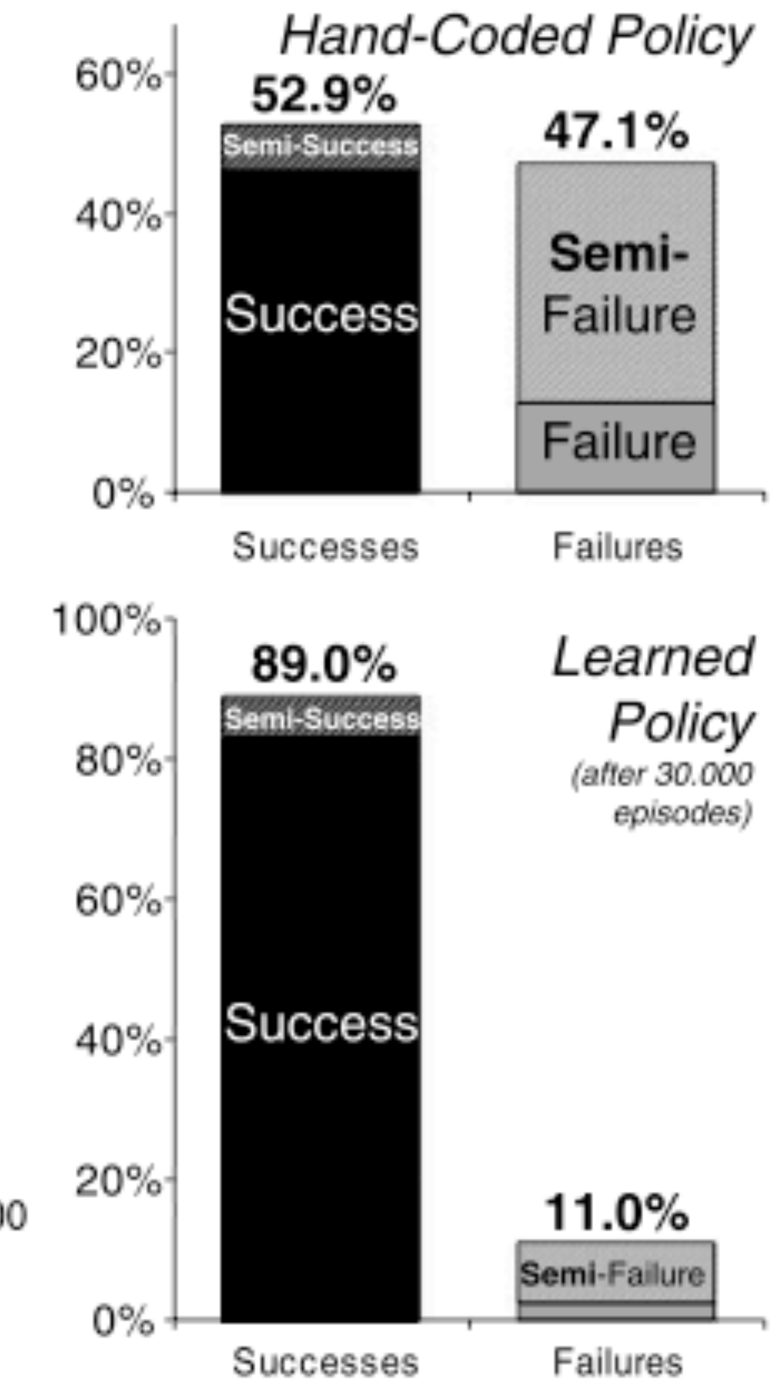
Learn new Q-function: $Q^{[k]}(\boldsymbol{s}, \boldsymbol{a}) \leftarrow \mathrm{Regress}(\boldsymbol{X}, \tilde{\boldsymbol{q}}^{[k]})$

end

➡ Like Value-Iteration, but we use supervised learning methods to approximate the Q-function at each iteration k

Ernst, Geurts and Wehenkel, *Tree-Based Batch Mode Reinforcement Learning, JMLR 2005*
Lange, Gabel and Riedmiller. *Batch Reinforcement Learning*, *Reinforcement Learning: State of the Art*

Learning Robot Soccer



Riedmiller et al., Reinforcement Learning in Robot Soccer, 2009

Value Function Methods

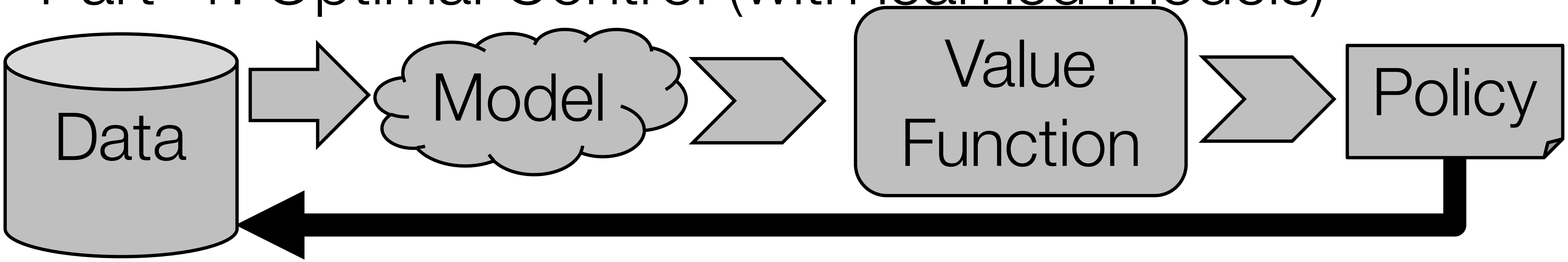
- ... have been the driving reinforcement learning approach in the 1990s.
- You can do loads of cool things with them: Learn Chess at professional level, learn **Backgammon and Checkers at Grandmaster-Level** ... and winning the **Robot Soccer Cup** with a minimum of man power.

So, why are they not always the method of choice?

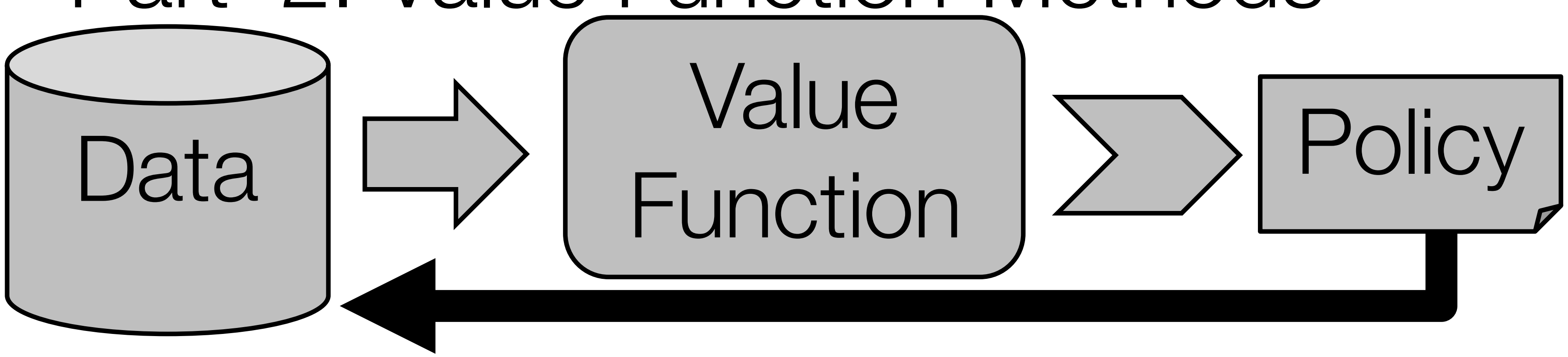
- You need to fill-up you state-action space up with sufficient samples.
- Another curse of dimensionality with an exponential explosion.
- Errors in the Value function approximation might have a catastrophic effect on the policy, can be very hard to control
- However, it scales better as we only need samples at relevant locations.

Reinforcement Learning

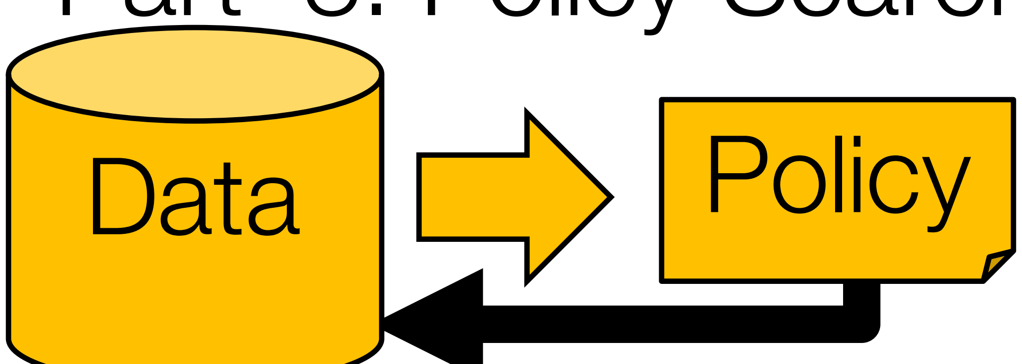
Part 1. Optimal Control (with learned models)



Part 2. Value Function Methods



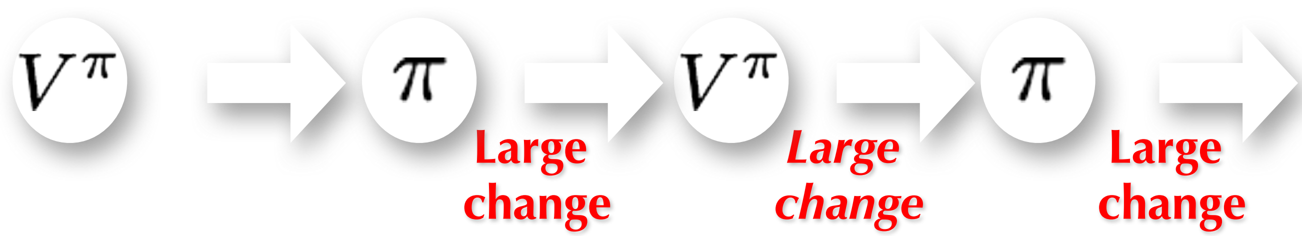
Part 3. Policy Search



Greedy vs Incremental

Greedy Updates:

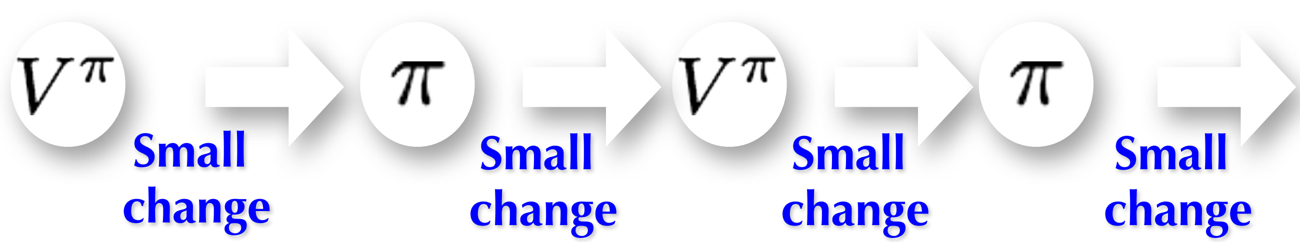
$$\boldsymbol{\theta}_{\pi'} = \text{argmax}_{\tilde{\boldsymbol{\theta}}} E_{\pi_{\tilde{\theta}}} \{Q^{\pi}(\boldsymbol{x}, \boldsymbol{u})\}$$



potentially unstable learning process with large policy jumps

Policy Gradient Updates:

$$\boldsymbol{\theta}_{\pi'} = \boldsymbol{\theta}_{\pi} + \alpha \left. \frac{dJ(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_\pi}$$



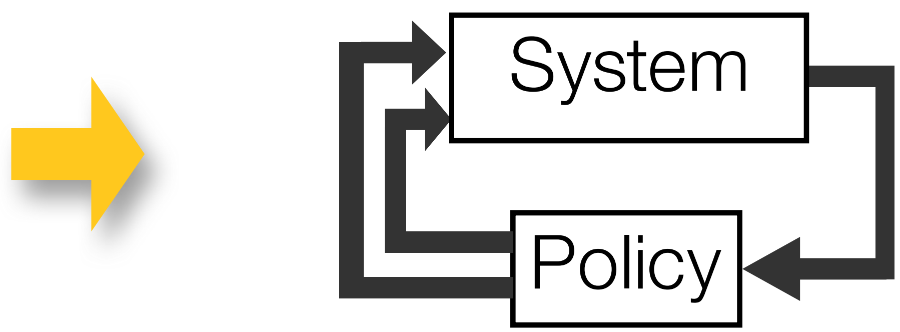
stable learning process with smooth policy improvement

Peters & Schaal, Reinforcement Learning of Motor Skills with Policy Gradients, 2008

Black-Box Approaches, e.g., Finite Differences

1. Perturb the parameters of your policy:

$\theta + \delta\theta$



$$\delta J = J(\boldsymbol{\theta} + \delta\boldsymbol{\theta}) - J(\boldsymbol{\theta})$$

2. Approximate J by first order Taylor approximation

$$J(\boldsymbol{\theta} + \delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \delta\boldsymbol{\theta}$$

3. Solve for $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ in a least squares sense (linear regression):

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{FD}} J = \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = (\Delta\boldsymbol{\Theta}^T \Delta\boldsymbol{\Theta})^{-1} \Delta\boldsymbol{\Theta}^T \Delta\boldsymbol{J}$$

A large class of algorithms includes Kiefer-Wolfowitz procedure, Robbins-Monroe, Simultaneous Perturbation Stochastic Approximation SPSA, ...

Likelihood-Ratio Policy Gradient methods

Some more basic notation

Trajectory distribution: $p(\boldsymbol{\tau};\boldsymbol{\theta}) = p(\boldsymbol{s}_1)\prod_{t=1}^{T-1}\pi(\boldsymbol{a}_t|\boldsymbol{s}_t;\boldsymbol{\theta})p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)$

Return for a single trajectory: $R(\boldsymbol{\tau}) = \sum_{t=1}^{T-1} r_t + r_T$

Expected long term reward $J(\boldsymbol{\theta})$ **can be written as expectation over the trajectory distribution**

$$J(\boldsymbol{\theta}) = \mathbb{E}_{p(\boldsymbol{\tau};\boldsymbol{\theta})}[R(\boldsymbol{\tau})] = \int p(\boldsymbol{\tau};\boldsymbol{\theta})R(\boldsymbol{\tau})d\tau$$

Peters & Schaal, Reinforcement Learning of Motor Skills with Policy Gradients, 2008

Likelihood Ratio Gradient

The step-based policy gradient can be computed efficiently **by the likelihood-ratio trick**

$$\nabla \log f(x) = \frac{1}{f(x)} \nabla f(x) \quad \Rightarrow \quad \nabla f(x) = f(x) \nabla \log f(x)$$

Applied to the policy gradient

$$\nabla_{\boldsymbol{\theta}} J = \nabla_{\boldsymbol{\theta}} \int p(\boldsymbol{\tau}; \boldsymbol{\theta}) R(\boldsymbol{\tau}) d\tau = \int \nabla_{\boldsymbol{\theta}} p(\boldsymbol{\tau}; \boldsymbol{\theta}) R(\boldsymbol{\tau}) d\tau$$

$$= \int p(\boldsymbol{\tau}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau}; \boldsymbol{\theta}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$\approx \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau}^{[i]}; \boldsymbol{\theta}) R(\boldsymbol{\tau}^{[i]})$$

Needs only samples!

Peters & Schaal, Reinforcement Learning of Motor Skills with Policy Gradients, 2008

Likelihood Ratio Gradient

How do we compute $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau}^{[i]}; \boldsymbol{\theta})$?

$$p(\boldsymbol{\tau}; \boldsymbol{\theta}) = p(\boldsymbol{s}_1) \prod_{t=1}^{T-1} \pi(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta}) p(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t)$$

The good old log again…

$$\log p(\boldsymbol{\tau}; \boldsymbol{\theta}) = \sum_{t=1}^{T-1} \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta}) + \mathrm{const}$$

Derivative is now easy…

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau}; \boldsymbol{\theta}) = \sum_{t=1}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta})$$

Peters & Schaal, Reinforcement Learning of Motor Skills with Policy Gradients, 2008

Lets plug it in…

Result:

$$\nabla_{\boldsymbol{\theta}} J = \sum_{i=1}^{N} \sum_{t=1}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}_t^{[i]} | \boldsymbol{s}_t^{[i]}; \boldsymbol{\theta}) R(\boldsymbol{\tau}^{[i]})$$

$$= \sum_{i=1}^{N} \sum_{t=1}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}_t^{[i]} | \boldsymbol{s}_t;^{[i]} \boldsymbol{\theta}) \left(\sum_{t=1}^{T-1} r_t^{[i]} + r_T^{[i]} \right)$$

This algorithm is called the REINFORCE Policy Gradient

Does this method work well?

No!

Peters & Schaal, Reinforcement Learning of Motor Skills with Policy Gradients, 2008

Kullback Leibler divergences

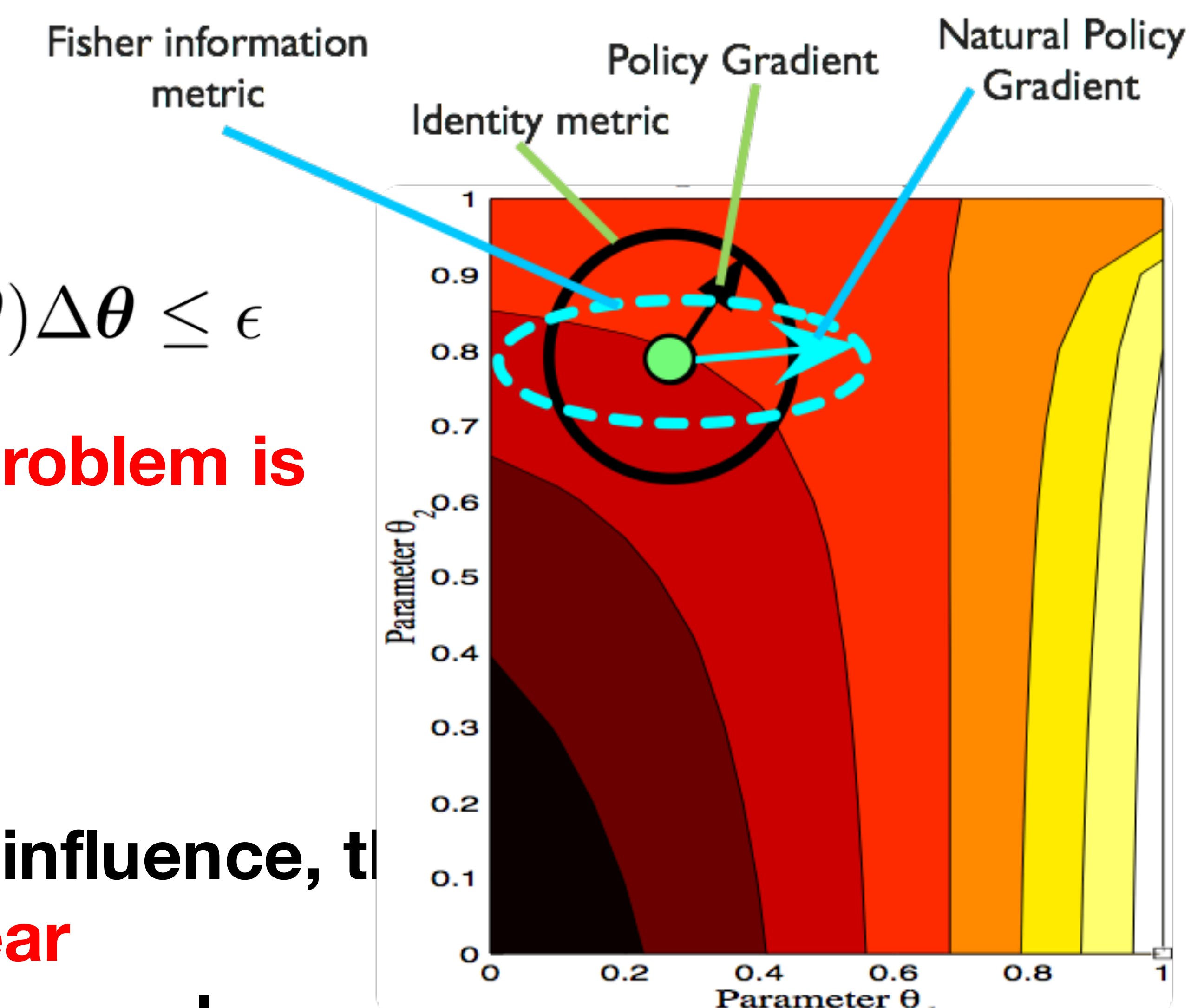
The Natural gradient is defined as the update direction which is closest to the standard gradient, but has limited distance to the old distribution

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}} J = \mathrm{argmax}_{\Delta\boldsymbol{\theta}} \Delta\boldsymbol{\theta}^T \nabla_{\boldsymbol{\theta}} J$$

$$\text{s.t.:} \quad \mathrm{KL}(p_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}} || p_{\boldsymbol{\theta}}) \approx \Delta\boldsymbol{\theta}^T \boldsymbol{G}(\boldsymbol{\theta}) \Delta\boldsymbol{\theta} \leq \epsilon$$

The solution to this optimization problem is given as:

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}} J \propto G(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} J$$



As every parameter has the same influence, tl natural gradient is invariant to linear transformations of the parameter space!

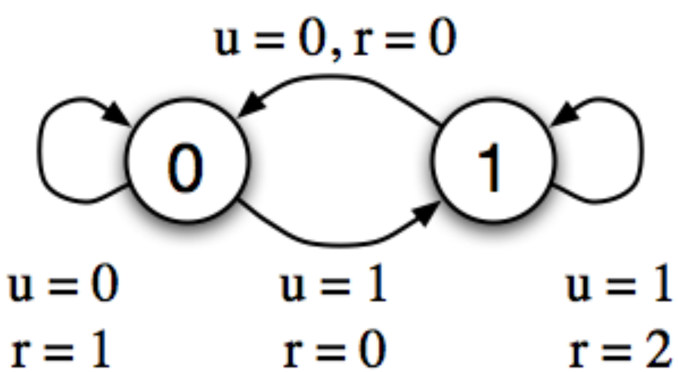
Kakade, 2002: Peters & Schaal, Natural-Actor Critic, 2003; Bagnell & Schneider, 2003

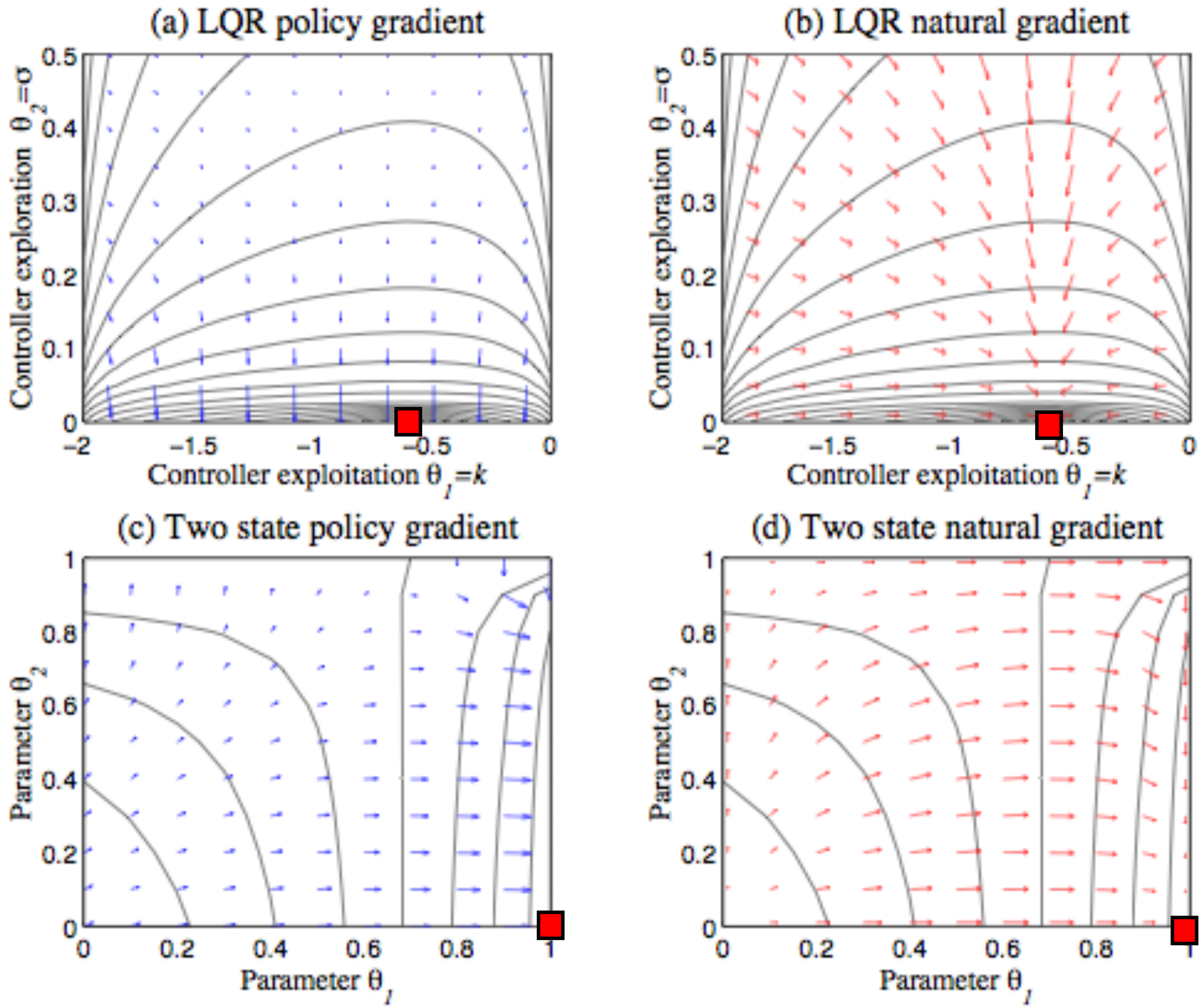
Are they useful?

Linear Quadratic Regulation

$$x_{t+1} = Ax_t + Bu_t$$
$$u_t \sim \pi(u|x_t) = \mathcal{N}(u|kx_t, \sigma)$$
$$r_t = -x_t^T Q x_t - u_t^T R u_t$$

Two-State Problem





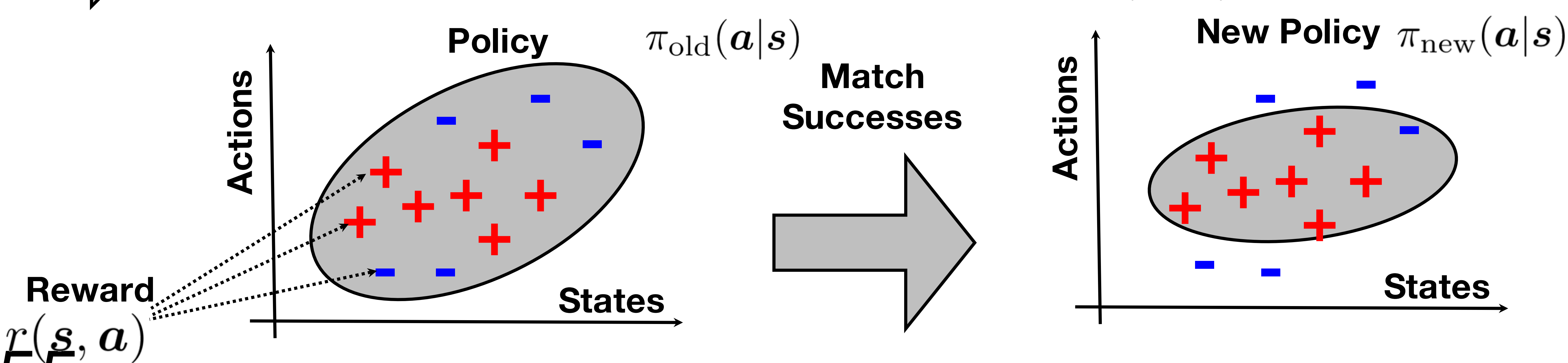
(Peters et al. 2003, 2005)

Success Matching Principle

> "When learning from a set of their own trials in iterated decision problems, humans attempt to match **not the best taken action** but the **reward-weighted frequency** of their actions and outcomes" (Arrow, 1958).

- Why? We still need to explore!
- Create policies such that $\pi_{\text{new}}(\boldsymbol{a}|\boldsymbol{s}) \propto \pi_{\text{old}}(\boldsymbol{a}|\boldsymbol{s}) r(\boldsymbol{s}, \boldsymbol{a})$

⇒ Only possible for non-negative reward functions $r(\boldsymbol{s}, \boldsymbol{a})$

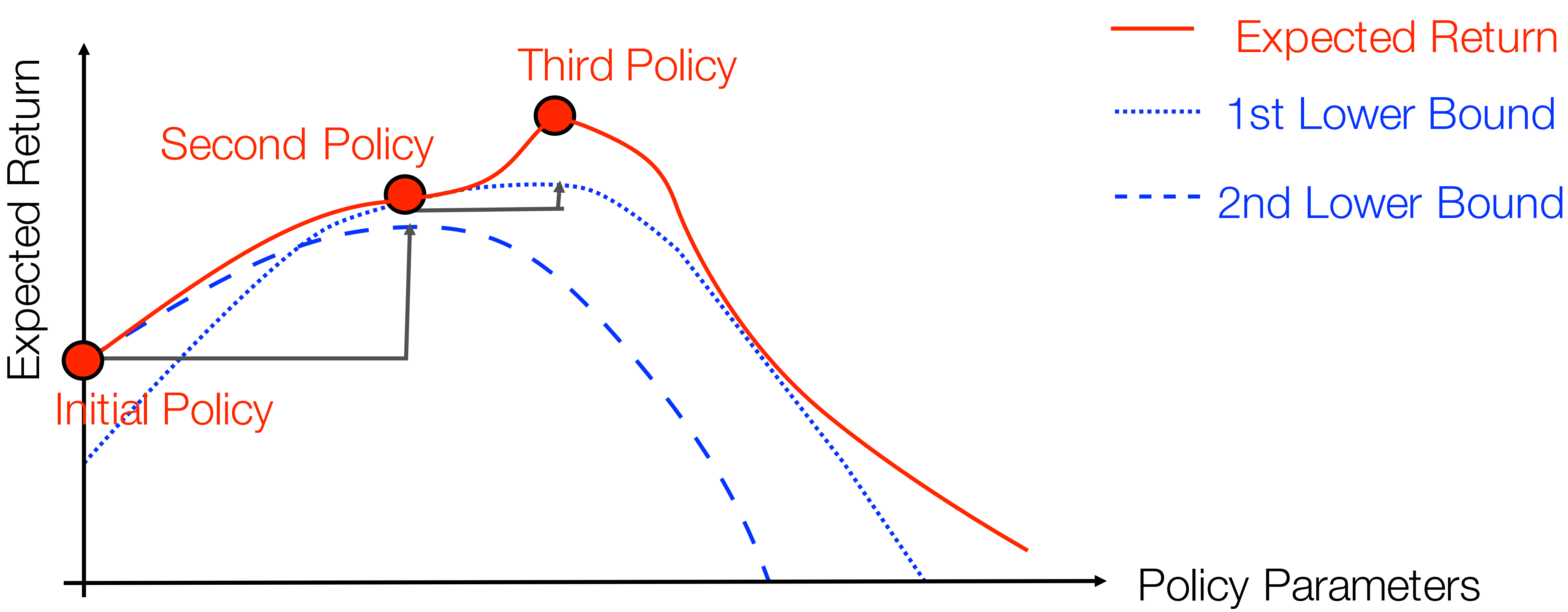


+ Succes (high reward) **- Failure (low reward)**

Basic Intuition

- Lower Bound on Expected Return
  - reward is an improper probability distribution
  - log-likelihood → log(expected return)
    (Dayan & Hinton, Neural Computation 1997; Peters & Schaal, ICML 2007; Kober & Peters, NIPS 2008)

$$\log J(\boldsymbol{\theta}') \geq \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\, R(\boldsymbol{\tau}) \log \frac{p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} d\boldsymbol{\tau} + \text{const} = L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$$

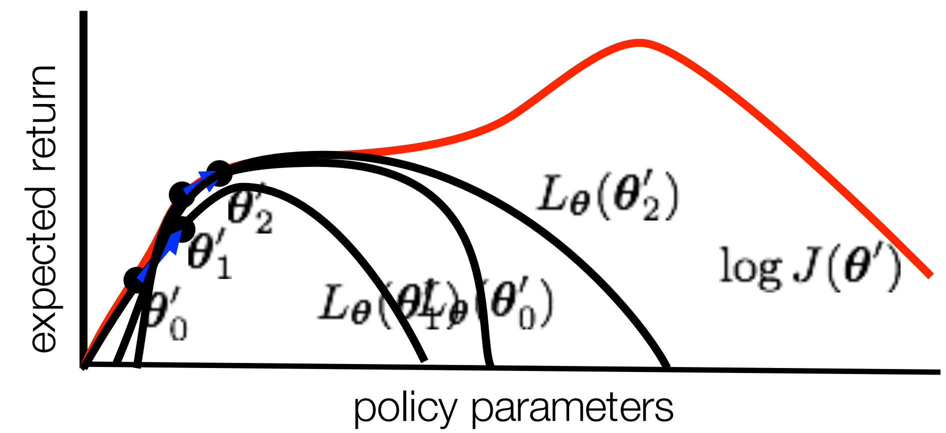


Resulting Algorithms

Policy Gradients: maximize lower bound by following the gradient

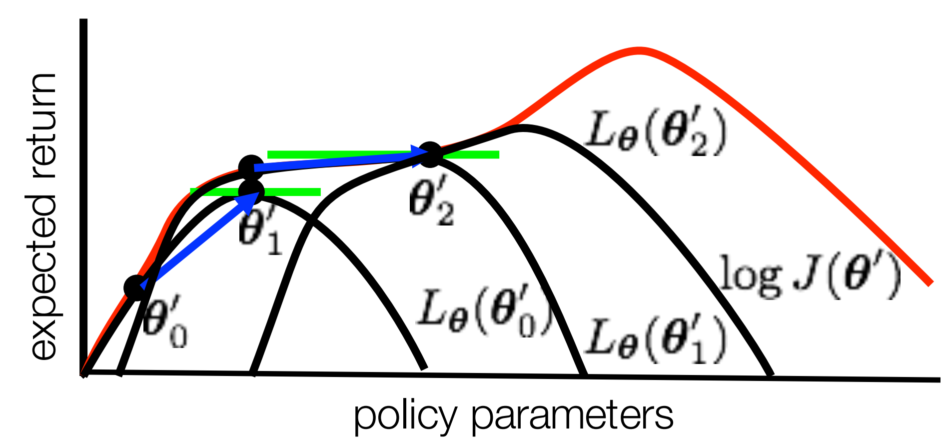
$$\lim_{\boldsymbol{\theta}' \to \boldsymbol{\theta}} \partial_{\boldsymbol{\theta}'} L_{\boldsymbol{\theta}}(\boldsymbol{\theta}') = \partial_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}' \approx \boldsymbol{\theta} + \alpha \partial_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$



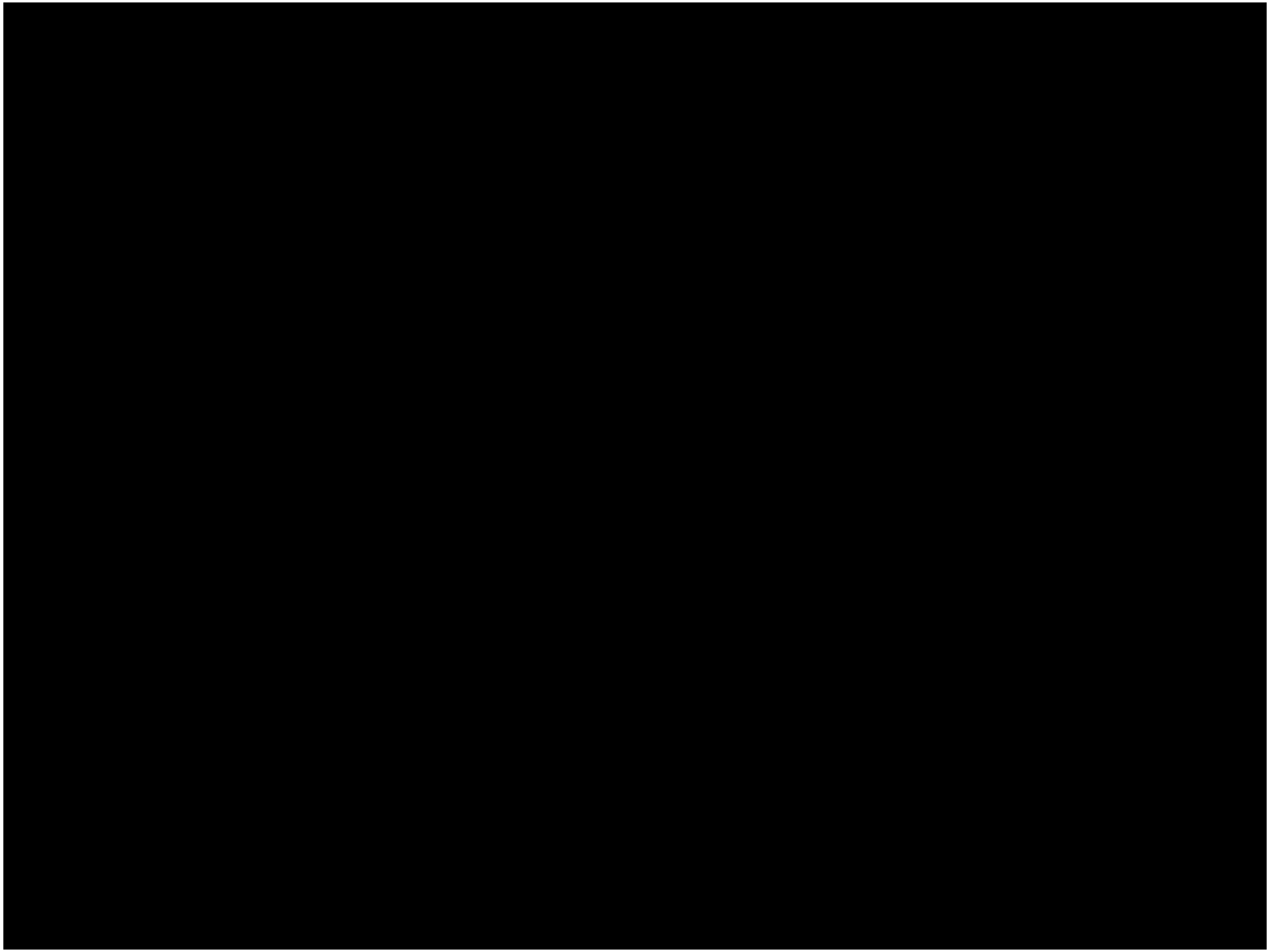
EM-like Methods: maximize lower bound by expectation-maximization

$$\boldsymbol{\theta}' = \operatorname{argmax} L_{\boldsymbol{\theta}}\left(\boldsymbol{\theta}'\right)$$



Kober & Peters, NIPS 2009+Machine Learning (MLJ) 2011

Ball in the Cup



58

What are the key problems?

Key Problems

1. no notion of data in the generic problem formulation

   Let's introduce the *observed data distribution* $q(\mathbf{x},\mathbf{u}))$.

2. optimization bias problematic with data

   Let's bound our information loss $D(\mu^{\pi_\theta}(\mathbf{x})\pi_\theta(\mathbf{u}|\mathbf{x})||q(\mathbf{x},\mathbf{u}))\leq\epsilon$

3. role of features is unclear in most methods

   Let's introduce stationary features determined by

$$\sum_{\mathbf{x}'}\mu^\pi(\mathbf{x}')\phi_{\mathbf{x}'}=\sum_{\mathbf{x},\mathbf{u},\mathbf{x}'}\mu^\pi(\mathbf{x})\pi(\mathbf{u}|\mathbf{x})p(\mathbf{x}'|\mathbf{x},\mathbf{u})\phi_{\mathbf{x}'}$$

Relative Entropy Policy Search Problem

Peters et al., AAAI 2010

$$\max_{\mu^\pi,\pi} J(\pi) = \sum_{\mathbf{x},\mathbf{u}} \mu^\pi(\mathbf{x})\pi(\mathbf{u}|\mathbf{x})r(\mathbf{x},\mathbf{u})$$

$$s.t.$$

$$\forall \mathbf{x}' : \mu^\pi(\mathbf{x}') = \sum_{\mathbf{x},\mathbf{u}} \mu^\pi(\mathbf{x})\pi(\mathbf{u}|\mathbf{x})p(\mathbf{x}'|\mathbf{x},\mathbf{u})$$

$$\sum_{\mathbf{x}'} \mu^\pi(\mathbf{x}')\phi_{\mathbf{x}'} = \sum_{\mathbf{x},\mathbf{u},\mathbf{x}'} \mu^\pi(\mathbf{x})\pi(\mathbf{u}|\mathbf{x})p(\mathbf{x}'|\mathbf{x},\mathbf{u})\phi_{\mathbf{x}'}$$

$$1 = \sum_{\mathbf{x},\mathbf{u}} \mu^\pi(\mathbf{x})\pi(\mathbf{u}|\mathbf{x})$$

$$\sum_{\mathbf{x},\mathbf{u}} \mu^\pi(\mathbf{x})\pi(\mathbf{u}|\mathbf{x}) \log \frac{\mu^\pi(\mathbf{x})\pi(\mathbf{u}|\mathbf{x})}{q(\mathbf{x},\mathbf{u})} \leq \epsilon$$

Solution

Peters et al., AAAI 2010

New Gibbs Policy

Value Function

$$\pi(\mathbf{u}|\mathbf{x}) = \frac{q(\mathbf{x},\mathbf{u}) \exp\left(\frac{1}{\eta}\delta_\theta(\mathbf{x},\mathbf{u})\right)}{\sum_{\mathbf{a}} q(\mathbf{x},\mathbf{a}) \exp\left(\frac{1}{\eta}\delta_\theta(\mathbf{x},\mathbf{a})\right)}$$

Advantage Function given as $V_\theta(\mathbf{x}) = \phi_{\mathbf{x}}^T \theta$

Parameters

$$\delta_\theta(\mathbf{x},\mathbf{u}) = r(\mathbf{x},\mathbf{u}) + \sum_{\mathbf{x}'} p(\mathbf{x}'|\mathbf{x},\mathbf{u}) V_\theta(\mathbf{x}') - V_\theta(\mathbf{x})$$

$$\max_{\theta,\eta} g(\theta,\eta) = \eta \log\left(\sum_{\mathbf{x},\mathbf{u}} q(\mathbf{x},\mathbf{u}) \exp\left(\epsilon + \frac{1}{\eta}\delta_\theta(\mathbf{x},\mathbf{u})\right)\right)$$

All direct results of the previous problem!

Wrap-Up

- Policy Search is a powerful and practical alternative to value function and model-based methods.

- Policy gradients have dominated this area for a long time and solidly working methods exist.

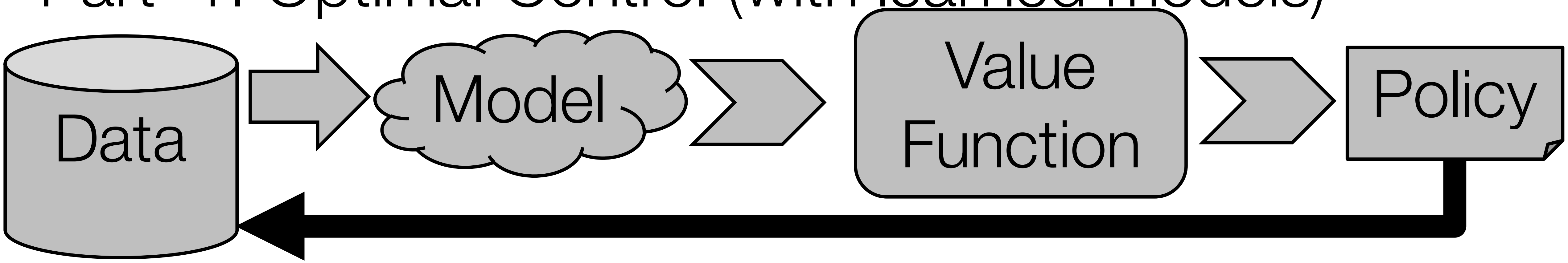
- Learning the exploration rate is still an open problem

- Newer methods focus on probabilistic policy search approaches.

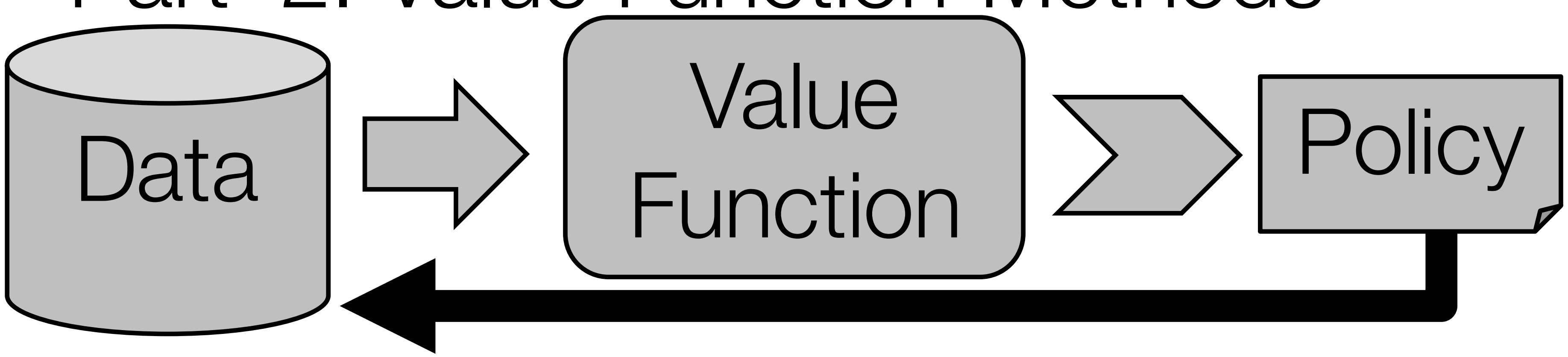
- Relative Entropy Policy Search (and its simplifications like TRPO) would be today's choice!

Reinforcement Learning

Part 1. Optimal Control (with learned models)



Part 2. Value Function Methods



Part 3. Policy Search